НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК 004:004.453

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
“___”_____2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 122 – комп’ютерні науки та інформаційні
(код і назва спеціальності)

технології (Системне проектування сервісів)

на тему _____База знань як сервіс_____

Виконала: студентка VI курсу, групи ДА-62
(шифр групи)

_____Слухай Яна Олександрівна_____

(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н. Булах Б.В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розробка стартап-проекту __к.т.н. Булах Б.В.
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 122 – комп'ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Слухай Яни Олександрівни
(прізвище, ім'я, по батькові)

1. Тема дисертації База знань як сервіс

науковий керівник дисертації ____к.т.н. Булах Богдан Вікторович_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації

3. Об'єкт дослідження: Бази знань

4. Предмет дослідження: знаходження ефективних конфігурацій розробки та хмарного розгортання веб-сервісу, що може взаємодіяти з базами знань у вигляді сховищ триплетів.

5. Перелік завдань, які потрібно розробити

- Проведення огляду концепцій та основних понять семантичної павутини

- Проведення порівняльного аналізу інструментів для створення бази знань у вигляді сховища триплетів
- Дослідження підходів до побудови веб-сервісів
- Формулювання вимог до системи «База знань як сервіс»
- Розробка та тестування додатку для вирішення поставленої задачі
- Хмарне розгортання розробленого додатку
- Розроблення стартап-проекту «База знань як сервіс»

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: резентація на тему «База знань як сервіс»

7. Орієнтовний перелік публікацій

Slukhai Y. Comparative analysis of software for building a knowledge base. International scientific journal "Internauka"/ Yana Slukhai// - 2018. – №8.

Slukhai Y. The knowledge base as a service architectural requirements research. International scientific journal "Internauka"/ Yana Slukhai// - 2018. – №8.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
	Отримання завдання	02.10.2017	
	Огляд літературних джерел	22.01.2018	
	Огляд концепцій семантичного вебу	12.02.2018	
	Дослідження підходів до побудови веб-сервісів	05.03.2018	
	Розробка програмного додатку	25.03.2018	
	Хмарне розгортання додатку	09.04.2018	
	Тестування додатку	04.05.2018	
	Отримання допуску до захисту та подача роботи в ДЕК	14.05.2018	

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

ЗМІСТ

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: База знань як сервіс

студенткою: Слухай Яною Олександрівною

Робота виконана на 95 сторінках, містить 29 ілюстрацій, 25 таблиць. При підготовці використовувалась література з 35 джерел.

Актуальність теми

Аналіз сильно пов’язаних даних часто являється невід’ємною частиною систем для обробки даних. Побудова ефективних рішень, які б легко інтегрувались в уже існуючі системи, є однією з задач розробників сьогодення. Оскільки системи типу SaaS є простими в користуванні та не потребують підтримки від кінцевих користувачів, було би ефективно застосувати хмарне розгортання для системи обробки бази знань.

Надання доступу до системи обробки знань дасть користувачам можливість інтегрувати її у свої вже існуючі додатки. Знаходження ефективних засобів для побудови веб-сервісу на основі системи обробки бази знань є актуальним як для обробки даних науковцями, так і для подальшого впровадження у існуючі бізнес-системи.

Мета та задачі дослідження

Метою даної роботи є аналіз можливості надання доступу до бази знань як до сервісу, вибір найефективніших концепцій, інструментів та фреймворків для імплементації такої системи. Для надання системі легкості розширення, необхідно проаналізувати задачі, які повинні вирішуватись, та сформулювати конкретні вимоги; після чого можна обрати підхід до проектування та існуючі шаблони проектування. Задачею дипломної роботи є створення системи, що зможе бути розгорнутою у хмарному середовищі, яка буде надавати основні можливості роботи з семантичними даними та матиме універсальний інтерфейс для подальшого використання у інших системах.

Рішення поставлених завдань та досягнуті результати

Було розроблено RESTful API, що надає можливість взаємодії зі сховищем триплетів. Даний веб-сервіс було розгорнуто у хмарному середовищі Heroku та протестовано за допомогою середовища для розробників API Postman. Основний функціонал додатку дозволяє завантажувати у базу знань цілі онтології, виконувати SPARQL запити на читання та оновлення, отримувати список класів онтологій та їх підкласів, перелік транзитивних відношень та список RDF суб'єктів та їх властивостей.

Об'єкт досліджень

Бази знань

Предмет досліджень

Ефективні конфігурації розробки та хмарного розгортання веб-сервісу, що може взаємодіяти з базами знань у вигляді сховищ триплетів.

Методи досліджень

Для вирішення проблеми в даній роботі використовуються методи аналізу і синтезу, системного аналізу, порівняння, логічного узагальнення результатів, проектування логічних структур даних.

Наукова новизна

Наукова новизна роботи полягає у створенні нових ефективних конфігурацій розробки та розгортання веб-сервісу, що може взаємодіяти з базами знань у вигляді сховищ триплетів та є зручним для подальшої підтримки та розширення.

Практичне значення одержаних результатів

Одержаний додаток може бути використаним як частина бізнес-систем, що працюють з даними у вигляді триплетів та потребують прошарок для роботи зі

сховищами даних. Також такий веб-сервіс може бути використаним для аналізу даних у вигляді онтологій науковцями.

Апробації результатів дисертації

Результати роботи дисертації було оприлюднено у міжнародному науковому журналі «Інтернаука», випуск №8 2018 року.

Публікації

Слухай Я. О. Порівняльний аналіз програмних засобів для побудови бази знань /Яна Слухай // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.

Слухай Я. О. Дослідження архітектурних вимог до бази знань як сервісу/ Яна Слухай // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.

Ключові слова

Сховище триплетів, семантичний веб, онтологія, веб-сервіс, Heroku, Spring Boot, Apache Jena TDB, REST.

РЕФЕРАТ

на магистерскую диссертацию

выполненную на тему: База знаний как сервис

студенткой: Слухай Яной Александровной

Работа выполнена на 95 страницах, содержит 25 илюстраций, 29 таблиц. При

подготовке использовалась литература с 35 источников.

Актуальность темы

Анализ сильно связанных данных часто является неотъемлемой частью систем для обработки данных. Построение эффективных решений, которые легко интегрировать в уже существующие системы, является одной из задач разработчиков сегодняшнего дня. Поскольку системы типа SaaS просты в использовании и не требуют поддержки конечных пользователей, было бы эффективно применить облачное развертывание для системы обработки базы знаний.

Предоставление доступа к системе обработки знаний даст пользователям возможность интегрировать ее в свои уже существующие приложения.

Нахождение эффективных средств для построения веб-сервиса на основе системы обработки базы знаний является актуальным как для обработки данных учеными, так и для дальнейшего внедрения в существующие бизнес-системы.

Цель и задачи исследования

Целью данной работы является анализ возможности предоставления доступа к базе знаний как к сервису, выбор эффективных концепций, инструментов и фреймворков для имплементации такой системы.

Для предоставления системе легкости расширения, необходимо проанализировать задачи, которые должны решаться, и сформулировать конкретные требования; после чего можно выбрать подход к проектированию и существующие шаблоны проектирования.

Задачей дипломной работы является создание системы, которая сможет быть развернутой в облачной среде, которая будет предоставлять основные возможности работы с семантическими данными и иметь универсальный интерфейс для дальнейшего использования в других системах.

Решение поставленных задач и достигнутых результатах

Был разработан RESTful API, что позволяет взаимодействия с хранилищем триплетов. Данный веб-сервис был развернут в облачной среде Heroku и протестировано с помощью среды для разработчиков API Postman. Основной функционал приложения позволяет загружать в базу знаний цели онтологии, выполнять SPARQL запросы на чтение и обновления, получать список классов онтологий и их подклассов, перечень транзитивных отношений и список RDF субъектов и их свойств.

Объект исследований

Базы знаний

Предмет исследований

Эффективные конфигурации разработки и облачного развертывания веб-сервиса, который может взаимодействовать с базами знаний в виде хранилищ триплетов.

Методы исследований

Для решения проблемы в данной работе используются методы анализа и синтеза, системного анализа, сравнения, логического обобщения результатов, проектирование логических структур данных.

Научная новизна

Научная новизна работы заключается в создании новых эффективных конфигураций разработки и развертывания веб-сервиса, который может взаимодействовать с базами знаний в виде хранилищ триплетов и является удобным для дальнейшей поддержки и расширения.

Практическое значение полученных результатов

Полученное приложение может быть использовано как часть бизнес-систем, работающих с данными в виде триплетов и требуют прослойку для работы с хранилищами данных. Также веб-сервис может быть использован для анализа данных в виде онтологий учеными.

Апробации результатов диссертации

Результаты работы диссертации были обнародованы в международном научном журнале «Интернаука», выпуск №8 2018 года.

Публикации

Слухай Я. А. Сравнительный анализ программных средств для построения базы знаний/Яна Слухай // Международный научный журнал "Интернаука". - 2018. - №8.

Слухай Я. А. Исследование архитектурных требований к базе знаний как сервису /Яна Слухай// Международный научный журнал "Интернаука". - 2018. - №8.

Ключевые слова

Хранилище триплетов, семантический веб, онтология, веб-сервис, Heroku, Spring Boot, Apache Jena TDB, REST.

ABSTRACT

on the master's thesis
on topic: Knowledge base as a service

student: Yana O. Slukhai

Work carried out on 95 pages containing 29 figures, 25 tables. The paper was written with references to 35 different sources.

Topicality

Strongly related data analysis is often a sufficient part of data processing systems. Building effective solutions that are easy to integrate into existing systems is one of the tasks of today's developers. Because SaaS-type systems are easy to use and do not require end-user support, it would be efficient to apply cloud-based deployment to a knowledge management system.

Providing access to the knowledge management system will allow users to integrate it into their existing applications. Finding effective tools for building a web-service based on the knowledge base processing system is relevant both for data processing by scientists and for further integration into existing business systems.

Purpose and objectives of the study

The purpose of this paper is to analyze the possibility of providing access to the knowledge base as a service, choosing effective concepts, tools and frameworks for implementing such a system.

To provide the system with ease of expansion, it is necessary to analyze the tasks that need to be addressed and formulate specific requirements; then you can choose the approach to design and existing design patterns.

The task of the thesis is to create a system that can be deployed in a cloud environment that will provide basic capabilities for working with semantic data and have a universal interface for further use in other systems.

Meeting the objectives and results achieved

A RESTful API was developed, which allows interaction with the triplestore. This web-service was deployed in the Heroku cloud environment and tested using the

Postman API Developer Environment. The main functionality of the application allows you to load the ontologys into the knowledge base, perform SPARQL requests for reading and updates, obtain a list of ontology classes and their subclasses, list transitive relations, and list the RDF of subjects and their properties.

Object of research

Knowledge bases

Subject of research

Effective development configuration and cloud deployment of a web service that can interact with knowledge bases in the form of triplestores.

Research Methods

To solve the problem, methods of analysis and synthesis, system analysis, comparison, logical generalization of results, design of logical data structures were used.

Scientific novelty

The scientific novelty of the work is to create new effective configurations for the development and deployment of a web service that can interact with knowledge bases in the form of triple storage and is convenient for further support and expansion.

The practical significance of the results

The resulting application can be used as part of business systems that work with data in the form of triples and require a layer to work with data stores. Also, a web service can be used to analyze data in the form of ontologies by scientists.

Approbation of dissertation results

The results of the thesis were published in the International Scietific Journal «Internauka». No. 8 of 2018.

Publications

Slukhai Y. Comparative analysis of software for building a knowledge base. International scientific journal "Internauka"/ Yana Slukhai// - 2018. – №8.

Slukhai Y. The knowledge base as a service architectural requirements research. International scientific journal "Internauka"/ Yana Slukhai// - 2018. – №8.

Keywords

Triplestore, semantic web, ontology, web-service, Heroku, Spring Boot, Apache Jena TDB, REST.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

DDD - Domain Driven Development

DTO - Data Transfer Object

OWL - Web Ontology Language

RDF - Resource Description Framework

SaaS - Software as a service

SOLID - single responsibility, open-closed, Liskov substitution, interface segregation, dependency inversion

SPARQL - SPARQL Protocol and RDF Query Language

ВСТУП

Зі стрімким розвитком IT-індустрії з’являється все більше технологій для обробки та збереження даних, адже це необхідно як для бізнес-користувачів, так і для наукових дослідників. Розуміння якостей та особливостей даних дає можливість розробникам чітко сформулювати вимоги до систем їх обробки та збереження, які допоможуть у подальшому виборі інструментів для проектування таких систем.

База знань (KB) - це технологія, яка використовується для зберігання складної структурованої та неструктурованої інформації, яка використовується комп'ютерною системою. Початкове використання цього терміну було пов'язане з експертними системами, які були першими системами, що базуються на знаннях.

Для побудови баз знань часто використовують онтології, які включають в себе представлення, формальне іменування та визначення категорій, властивостей та відносин понять, даних та об'єктів, що обґрунтовують один чи багато доменів інформації. Онтології являють собою одне з ключових понять семантичної павутини - нової концепції розвитку Всесвітньої павутини і мережі Інтернет, яка створена і впроваджується Консорціумом Всесвітньої павутини.

Аналіз сильно пов’язаних неструктурованих даних часто являється невід’ємною частиною систем для обробки даних. Побудова ефективних рішень, які б легко інтегрувались в уже існуючі системи, є однією з задач розробників сьогодення. Оскільки системи типу SaaS є простими в користуванні та не потребують підтримки від кінцевих користувачів[10], було би цікаво скомбінувати хмарне розгортання та системи обробки баз знань.

Надання доступу до системи обробки знань дасть користувачам можливість інтегрувати її у свої вже існуючі додатки. Одним із способів такого розгортання додатку є розробка його у вигляді веб-сервісу.

Отже, метою даної роботи є аналіз можливості надання доступу до бази знань як до сервісу, вибір найефективніших концепцій, інструментів та

фреймворків для імплементації такої системи. Для надання системі якостей такої, яку легко розширювати та підтримувати у майбутньому, необхідно проаналізувати задачі, які повинні вирішуватись, та сформулювати конкретні вимоги. Знаючи чіткі цілі системи, є можливість підбору існуючих шаблонів проектування та підходів до побудови архітектури для їх досягнення.

Не менш важливим є вибір інструментів для роботи з даними у вигляді онтологій, тому для розробки бази знань як сервісу необхідно провести детальний порівняльний аналіз засобів для обробки та збереження триплетів.

Звісно, однією з ключових стадій розробки системи є її скрупульозне тестування, тому ця стадія повинна бути детально розглянутою у роботі.

1 ОГЛЯД КОНЦЕПЦІЙ СЕМАНТИЧНОЇ ПАВУТИНИ

1.1 Основні поняття структури семантичної павутини

Семантичною павутиною називають нову концепцію розвитку Всесвітньої павутини і мережі Інтернет. Вона була створена і впроваджується Консорціумом Всесвітньої павутини. Її також часто називають семантичним вебом або ж семантичною мережею. Взагалі, поняття семантичної мережі виникло раніше, воно ж і породило уже інше поняття - семантичну павутину. Тому ці, на перший погляд схожі визначення, варто відрізняти.

Отже, семантична мережа являє собою інформаційну модель предметної області. Вона має вигляд орієнтованого графа, вершини якого відповідають об'єктам предметної області, а ребра задають відносини між ними. У вигляді об’єктів можна представити найрізноманітніші процеси, предмети, структури, факти [5]. Таким чином, семантична мережа - один із видів представлення знань.

Як зазначено у статті The Semantic Web revisited [27], “у назві сполучені терміни з двох наук: семантика у мовознавстві вивчає сенс одиниць мови, а мережа в математиці є різновидом графу — набору вершин, сполучених дугами (ребрами)”. У семантичній мережі роль вершин виконують поняття бази знань, а дуги (причому направлені) задають відношення між ними. Отже, семантична мережа відображає семантику предметної області у вигляді понять і відношень між ними.

Отже, Семантичний Web можна представити як симбіоз двох напрямів, один з яких описує мови представлення даних[18]. На сьогоднішній день основними такими мовами є Розширювана Мова Розмітки XML (eXtensible Markup Language) і мова опису ресурсів RDF (Resource Description Framework). Є такоє і інші формати представлення семантичних даних, проте XML і RDF надають більше можливостей, тому саме їх рекомендують W3C. Другий напрям несе в собі теоретичне уявлення про моделі наочних областей. Такі моделі

наочних областей в термінології Семантичного Web називаються онтологіями[5]. 10 лютого 2004 року консорціумом W3C була затверджена і опублікована специфікація мови мережевих онтологій OWL (Web Ontology Language). Таким чином, дві галузі Семантичного Web використовують три ключові мови (відповідно, технологій)[18]:

- специфікацію XML для визначення синтаксису і структури документів,
- мову опису ресурсів RDF для визначення моделі кодування понять у онтологіях,
- мову опису онтологій OWL для визначення понять і відношень між ними.

Семантичний Web використовує також і інші мови, технології і концепції, зокрема, універсальні ідентифікатори ресурсів URI, цифрові підписи, системи логічного виведення і т.д [27].

Винахідник та засновник поняття “семантична павутина“ - Тім Бернерс-Лі [5]. У 2001 році він опублікував статтю «Наступний крок у розвитку Всесвітньої павутини». Саме його ідеєю було побудувати додаткові структури для існуючої мережі Інтернет, які б були здатними перетворити дані у павутині так, щоб їх могли ефективно оброблювати комп’ютери. Для реалізації такого задуму було впроваджено стандарт RDF (Resource Definition Framework) та RDFS (RDF Schema).

1.2 Бази знань

База знань (БЗ) - це технологія, яка використовується для зберігання складної структурованої та неструктурованої інформації, яка використовується комп'ютерною системою. Початкове використання цього терміну було пов'язане з експертними системами, які були першими системами, що базуються на знаннях[16].

Первинне використання терміна бази знань полягало в описі однієї з двох підсистем системи, основаної на знаннях. Система, основана на знаннях,

складається з бази знань, що представляє факти про світ та механізму висновків, який може виводити інші факти та, використовуючи певні правила чи інші форми логіки, виявляти нові факти або висвітлювати невідповідності[19].

Термін "база знань" був створений для того, щоб відрізняти цю форму знань від більш загального та широко використовуваного терміну бази даних. У той час (1970-ті роки) практично всі великі системи управління інформаційними системами зберігали свої дані в певному типі ієрархічної або реляційної бази даних. На цьому етапі в історії інформаційних технологій різниця між базою даних та базою знань була чіткою та однозначною.

База даних мала такі властивості[3]:

Наявність плоских даних. Дані, як правило, представлені у вигляді таблиць у строковому або чисельному форматах.

Підтримка декількох користувачів. Згідно конвенції, база даних повинна підтримувати можливість використовування її певних ресурсів декількома користувачами або системами одночасно.

Підтримка транзакцій. Важливою вимогою для бази даних було збереження цілісності даних та відповідності між ними при одночасному доступі користувачами. Це так звані властивості ACID: атомарність, узгодженість, ізольованість, довговічність.

Наявність довговічних даних. Корпоративна база даних потрібна для підтримки не тільки тисяч, але й сотень тисяч або більше рядків даних. Така база даних, як правило, потрібна для того, щоб зберігати конкретні цілі будь-якої індивідуальної програми; потрібно було зберігати дані на багато років і десятиліть, а не на час життя програми.

Перші системи, що базуються на знаннях, мали потреби в даних, які які мали протилежні вимоги. Експертна система вимагає структурованих даних не просто у вигляді таблиць з числами та символами, а даних з наявністю вказівників на інші об'єкти, які у свою чергу мають додаткові вказівники. Ідеальним

представленням бази знань є об'єктна модель (часто називається онтологією в літературі штучного інтелекту) з наявністю класів, підкласів та екземплярів[19].

Ранні експертні системи не мали необхідності обслуговувати багато користувачів одночасно чи реалізовувати механізми забезпечення транзактивності даних. Дані для ранніх експертних систем використовувались для отримання конкретної відповіді, наприклад, медичного діагнозу чи структури молекули.

Вимоги до обсягу та змісту даних також були різними для бази знань порівняно зі звичайною базою даних. База знань повинна містити факти про світ. Наприклад, щоб представити твердження про те, що "всі люди смертні". База даних, як правило, не повинна відображати загальні поняття, але натомість могла би бути використаною для збереження інформації у тисячах таблиць, що містять інформацію про конкретних людей[16].

1.3 RDF, RDFS

RDF специфікація визначає необхідним створення деяких ресурсів, для яких треба вказати зв'язки типу значення - властивість. Можна сказати, що RDF семантика повинна визначати онтологію конкретної предметної області. Згідно з [34] “саме онтологія з кожним роком завойовує все ширше і ширше застосування у розв'язку задачі представлення знань, семантичної адаптації інформаційних ресурсів та їх пошуку. Дана віха у розвитку специфікації концептуалізації предметної області є досить перспективною, також з її допомогою здійснюються спроби представлення ієрархічних понять”.

Ресурс в RDF -це будь-яка інформаційна або неінформаційна сутність. Як описано в [24] “утвердження, висловлене про ресурс, має вигляд "суб'єкт - предикат - об'єкт" і називається триплетом”. Твердження “сонце круглої форми” в RDF-термінології можна представити наступним чином: суб'єкт - “сонце”, предикат - “має форму”, об'єкт - “круглий”. Для позначення суб'єктів, відносин і об'єктів у RDF використовуються URI.

Множину RDF-тверджень можна представити у вигляді орієнтованого графу, в якому суб'єкти та об'єкти слугують вершинами, а ребра відображають відносини.

RDF сам по собі є не форматом файлу, а тільки лише абстрактною моделлю [4] даних, тобто описує пропоновану структуру, способи обробки та інтерпретації даних. Для зберігання і передачі інформації, покладеної в модель RDF, існує цілий ряд форматів запису.

Для обробки RDF-даних можна використати мови запитів: SPARQL (стандарт W3C), RQL, RDQL[35].

RDFS (англ. RDF Schema, «схема» RDF), також RDF / S, RDF-S, RDF (S) - набір класів та властивостей моделі представлення знань RDF, що є основою для опису онтологій з використанням розширеного RDF-словника для структури RDF-ресурсів. RDFS використовує кодування у вигляді RDF, тому такі триплети можуть зберігатися, оброблятися і запитуватися подібно до опису RDF-ресурсів, наприклад, за допомогою SPARQL[25].

Конструкції RDFS побудовані на основі RDF-словника і включаються в себе класи, властивості і допоміжні властивості (utility properties). Таким чином, RDF може висловити відношення між класами (клас-підклас) і властивостями (властивість-підвластивість, що в свою чергу дозволяє складати більш гнучкі запити для отримання інформації [24].

В описах нижче RDF-триплет вважається таким, що складається з ресурсу-суб'єкта, ресурсу-предиката і ресурсу-об'єкта.

RDFS класи:

- **rdfs: Resource** - клас, що включає всі ресурси, тобто, все, що описує RDF.
- **rdfs: Class** - описує що ресурс є класом для інших ресурсів. Визначення може бути рекурсивним. Для того, щоб віднести ресурс до конкретного типу, використовується властивість **rdf: type**.

- **rdfs: Literal** - позначає літерал, наприклад, рядок або ціле число. Літерали можуть бути простими (plain) або мати певний тип.
- **rdfs: Datatype** - клас типів даних. Є одночасно і підклассом rdfs: Class, і екземпляром з rdfs: Class. Кожен екземпляр класу rdfs: Datatype є підкласом rdfs: Literal.

RDF класи:

- **rdf: XMLLiteral** - клас XML-литералов, є екземпляром rdfs: Datatype.
- **rdf: Property** - клас властивостей.
- **rdf: Statement** - клас тверджень RDF
- **rdf: List** - клас списків RDF
- **rdf: nil** - екземпляр rdf: список, що представляє порожній список

RDFS властивості

Властивості описують відношення між ресурсами-суб'єктами і ресурсами-об'єктами і є екземплярами класу **rdf: Property**[24]. При використанні в якості предиката в триплеті:

- **rdfs: domain** оголошує клас суб'єкта.
- **rdfs: range** оголошує клас або тип даних об'єкта.
- **rdfs: subClassOf** - властивість, що дозволяє описати ієрархію класів.
- **rdfs: subPropertyOf** - властивість, яке стверджує, що всі ресурси, пов'язані деяким подсвойством (subproperty), пов'язані також і властивістю.
- r**dfs: label** і **rdfs: comment** задають зручні для людини ім'я та опис ресурсу.
- **rdfs: seeAlso** вказує ресурс, який може служити джерелом додаткової інформації про ресурс-суб'єкт.
- **rdfs: isDefinedBy** вказує на ресурс (наприклад, RDF-словник), в якому описується ресурс-суб'єкт.

Наприклад, наступний набір RDF-триплетів характеризує ex: employer (роботодавця) як зв'язуючого відношенням особистість і організацію. З такого

набору випливає, що ex: OrganizationX є організацією, а ex: Yana - особистістю, в сенсі, який вкладає в ці поняття FOAF:

ex: employer rdfs: domain foaf: Person

ex: employer rdfs: range foaf: Organization

ex: Yana ex: employer ex: OrganizationX

У наступному прикладі стверджується, що «кожна особистість є агентом» (в сенсі FOAF):

foaf: Person rdfs: subClassOf foaf: Agent

Ієрархія класів підтримує успадкування домену (domain) і безлічі значень (range) від класу до підкласу.

RDF властивості

- **rdf: type** декларує приналежність ресурсу певному класу, тобто, той факт, що ресурс є екземпляром класу. Зазвичай для цього властивості використовується уточнене ім'я.
- **rdf: first** - перший елемент у списку RDF предметів
- **rdf: rest** - решта частини RDF списку після rdf: спочатку
- **rdf: value** - idiomatic властивість, що використовується для структурованих значень
- **rdf: subject** - предмет тематики RDF
- **rdf: predicate** - предикат суб'єкта RDF твердження
- **rdf: object** - об'єкт суб'єкта тесту RDF

Можливе використання цих властивостей залежить від конкретних додатків, інтерпретуючих RDFS. Наприклад, посилання на ресурси може бути приведена на веб-сторінці, що згенерована на основі RDF.

Перша версія RDF Schema була опублікована W3C в квітні 1998 року, а остаточна рекомендація [24] - в лютому 2004 року. На 2014 існує рекомендація для RDF Schema версії 1.1 [24]. Велика кількість компонентів RDF включені в більш виразну мову опису онтологій OWL.

1.4 OWL

Мова веб-онтологій OWL(Web Ontology Language) - це мова для визначення онтологій в мережі Інтернет. Онтологія OWL описує домен з точки зору класів, властивостей та окремих індивідів і може включати в себе багатий опис характеристик цих об'єктів. OWL онтології можуть бути використані для опису властивостей веб-ресурсів. Якщо раніше мови представлення були використані для розробки інструментів та онтологій для певних спільнот користувачів у таких галузях, як наука, охорона здоров'я та електронна комерція, то вони не завжди були розроблені таким чином, щоб бути сумісними з Всесвітньою мережею, а точніше з Семантичною мережею, як і у випадку з OWL.

Онтології є універсальними і ефективними засобами репрезентації та збереження даних, по суті, базами знань[8].

Онтологія представляє собою ієрархію понять зв’язаних відношенням деяких спеціальних видів. Дані онтології є рикладом семантичних мереж, які задаються у вигляді орієнтовних графів, у яких вершини означають поняття або їх властивості, а ребра – відношення деяких типів. До подібних відношень можна віднести такі: «належить», «є наслідком» і деякі інших[8].

Більш складні онтології формалізуються засобами мов логіки і допускають можливість логічного виводу. У найпростішому випадку онтології використовуються для підвищення точності пошуку в Інтернеті. Як було описано у [13] “також, якщо одні і ті ж поняття представляються різними термінами, механізм онтологій дозволяє формувати осмислені ієрархічні зв’язки між об’єктами, узагальнювати різні дані, реалізувати нечіткий пошук”.

Онтології були створені для виявлення певними складними інформаційними системами конкретних структур знань та певних логічних правил, що могли би бути використаними для виявлення нових фактів у конntексті таких структур.

Формальна семантика мови OWL, яка рекомендована W3C, описує як отримати логічні висновки на основі онтологій, тобто, отримати факти, які не представлені буквально, а випливають із семантики онтологій. При чому ці виводи можуть будуватися на аналізі як одного документа, так і множити документів, розподілених у мережі[8].

W3C вказує, що логічні висновки можна отримати за допомогою онтологій, які не зазначені у описі буквально, а є певним результатом застосування правил виведення, які теж є описаними у даному стандарті. Також цікавим є те, що для аналізу може бути представлене не одне, а декілька джерел, наданих у вигляді онтологій.

1.4.1 Різновиди мови

- OWL Lite - це підклас OWL DL, який підтримує лише підмножину конструкцій мови OWL. OWL Lite особливо орієнтована на розробників, які хочуть підтримувати OWL, але хочуть почати з порівняно простих основних наборів функцій мови. OWL Lite дотримується тих самих семантичних обмежень, що й OWL DL, що дозволяє гарантувати певні бажані властивості[13].
- OWL Full і OWL DL підтримують один і той же набір конструкцій мови OWL. Їхні відмінності полягають у обмеженні використання деяких з цих функцій та використання функцій RDF. OWL Full дозволяє вільно змішувати OWL з RDF Schema, і, як і RDF Schema, не забезпечує чіткого поділу класів, властивостей, окремих осіб і значень даних. OWL DL ставить обмеження на змішування з RDF і вимагає незмінність класів, властивостей, окремих осіб і значень даних. Основна причина наявності підрівня OWL DL полягає в тому, що розробники розробили потужні системи логічного виведення, які підтримують онтології, що обмежені обмеженнями, необхідними для OWL DL.

1.4.2 Структура мови

Онтології OWL зазвичай містять у собі такі основні елементи:

- індивіди,
- класи,
- властивості.

Індивіди(Individuals). Індивіди є конкретними об’єктами деякої доменної області. Також є цікавим той факт, що на один об’єкт можуть посилатись різні індивіди, як на такий, який вони представляють, якщо у правилах дескриптивної логіки не прописати явне правило, яке б це забороняло.

Властивості (Properties). Можна стверджувати, що властивості поєднують кількох індивідів. Являють собою бінарні взаємовідношення.

Класи (Classes). Це множини, елементами яких є індивіди. Вони описуються, використовуючи формальні (математичні) конструкції, які декларують вимоги для прийняття участі в класі[15]. Класи можна організувати в деяку ієрархію відношень виду «підкласс-суперклас», яка відома як таксономія. Підкласи є підмножинами свого суперкласу. Одна з головних особливостей OWL-DL - те, що ці відношення підкласу і суперкласу (відношення категоризації) можуть бути обчислені автоматично за допомогою різонерів [13].

1.5 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) — мова запитів до даних, що наявні у певній моделі RDF, а також протокол для передачі цих запитів і відповідей на них[6]. SPARQL рекомендується консорціумом W3C як стандартна мова запитів до онтологій і виступає однією з ключових технологій семантичної мережі. Надання SPARQL-точок доступу є рекомендованою практикою при розташуванні даних у мережі Інтернет.

Загальний вигляд SPARQL-запиту має такий вигляд[31]:

```
PREFIX foo: <http://example.com/resources/>
# префікси
FROM ...
# джерела
SELECT ...
# вибірка
WHERE {...}
# критерії запиту
ORDER BY ...
# модифікатори запиту
```

- Префікси використовуються для подальшого скорочення URI.
- Джерела запиту визначають, які RDF ресурси будуть брати участь у запиті.
- Вибірка позначає певний масив даних, який необхідно повернути користувачеві за його критеріями.
- Критерії запиту визначають обмеження, за якими необхідно профільтрувати кінцеву вибірку
- Модифікатори запиту допомагають згрупувати, відсортувати або ще якимось чином впорядкувати дані.

SPAQRL даж можливість користувачеві задавати однозначні запити. Наприклад, наступний запит повертає прізвища і адреси людей:

```
PREFIX example : <http://emample/0.1/>
SELECT ?surname
     ?ardess
WHERE {
 ?person  a         example:Person .
 ?person  example:surname  ?surname .
 ?person  example:ardess  ?ardess .
}
```

1.6 Логічне виведення

Семантичний різонер - це програмне забезпечення, що за допомогою певних правил та фактів здатне виконувати логічний аналіз для виведення додаткових, явно не прописаних у онтологіях залежностей. Такі правила можуть бути заданими за допомогою тих же мов, які описують онтології або ж за допомогою описової логіки. Різонери є дуже корисним інструментом для обробки великих онтологій, особливо таких, що використовують діалект OWL DL. Також надає функціонал для вирішення задач класифікації.

В OWL різонери працюють на основі концепції OWR – «Open World Reasoning». Open World Reasoning буквально перекладається з англійської як «міркування про відкритість світу». В OWL DL - це категорія, що позначає відкритість баз знань. Вона використовує концепцію відкритості світу, що принципово відрізняє бази знань від баз даних, заснованих на припущенні про замкнутість світу [4]. По суті це означає неповноту знань по відношенню до фактів, які можна вивести на їх основі шляхом логічних міркувань. Тобто побудова повної класифікації за описами окремих класів, які здавалося б, не дають повного опису онтології. Отже дані, які не були задані явно, але є такими, що виводяться за допомогою правил, можна теж вважати валідними для опису предметної області. Можливість наявності деяких припущень у предметній області дає можливість легко змінити ці припущення при зміні наших знань про предметну область[33]. Жорстке кодування припущень про світ на мові програмування призводить до того, що ці припущення не тільки складно знайти і зрозуміти, але і також складно змінити, особливо людям нетехнічного профілю [4]. Без розуміння принципів побудови онтологій, робота різонера може здаватися непомітною.

1.6.1 Основні задачі різонерів

Можна назвати такі основні задачі різонерів:

- Класифікація і виведення ієрархій класів.
- Перевірка онтологій на консистентність.
- Знаходження транзитивних відношень.
- Визначення належності індивіда до якогось класу.
- Визначення класів, що не є такими, які перетинаються з заданим класом.
- Знаходження підкласів певних класів

1.7 Сховища триплетів

Сховище триплетів або RDF-сховище є базою даних для зберігання та вилучення триплетів через семантичні запити. Триплет - це об'єкт даних, що складається з суб’єкта, об'єкта і предиката, наприклад "Ділан любить борщ" або "Ділан знає Джорджа".

На відміну від реляційної бази даних, сховище триплетів створене та оптимізоване для зберігання та вилучення триплетів. Окрім запитів, зазвичай можна імпортувати / експортувати триплети, використовуючи структуру опису ресурсів (RDF) та інші формати[14].

Деякі сховища триплетів були побудовані як бази знань з нуля, в той час як інші були побудовані на основі існуючих комерційних реляційних баз даних (наприклад, на базі SQL) або на основі баз даних, орієнтованих на документи [NoSQL]. Подібно ранньому розвитку баз даних оналайн аналітичної обробки (OLAP), цей проміжний підхід дозволив побудувати великі та потужні бази знань, що потребували невеликих зусиль для програмування на початкових етапах розробки сховищ триплетів[29].

Найбільш ймовірно, що нативні сховища триплетів матимуть перевагу для роботи протягом більш тривалого періоду часу. Труднощі з реалізацією сховищ триплетів на основі SQL полягає в тому, що, хоча триплети, таким чином, можуть бути "збережені", реалізація ефективного запиту до графічної RDF-моделі (наприклад, відображення з SPARQL) на SQL-запити є складною[14].

1.8 Аналіз фреймворків та інструментів для роботи з семантичною павутиною

Далі буде наведено результати попереднього дослідження, що були опубліковані у статті “Comparative analysis of software for building a knowledge base”.

1.8.1 Apache Jena - TDB

Apache Jena - це Java-фреймворк для побудови семантичних веб-додатків. Він забезпечує програмне середовище для RDF, RDFS і OWL, SPARQL, і включає в себе механізм побудови висновків на основі правил (підтримка роботи різонерів).

Apache Jena TDB надає легке, масштабовне, транзакційне сховище та можливість виконання SPARQL запитів.

Сховище TDB може бути доступним та керованим за допомогою скриптів командного рядка та через Jena API. При доступі за допомогою транзакцій TDB набір даних захищений від пошкоджень, несподіваних завершень процесу та системних збоїв[7].

Якщо є необхідність поділитися набором даних TDB між кількома додатками, можна використовувати компонент Fuseki, який надає сервер SPARQL. Fuseki може використовувати TDB для постійного зберігання, і надає протоколи SPARQL для запиту та оновлення згідно REST через HTTP[28].

1.8.2 Dydra

Завдяки Dydra можна отримати доступ до своїх даних і оновлювати їх за допомогою стандартної мови запитів, спеціально розробленої для обробки графів SPARQL.

Користувачі можуть виконувати більшість функцій облікового запису на Dydra.com за допомогою стандартного REST API[21]. Деякі з функцій включають: читання та оновлення інформації про обліковий запис; створення сховища; завантаження в сховище в різних форматах RDF. Відповіді надаються у форматі XML або JSON[23].

1.8.3 Sesame

Sesame - це сховище RDF із відкритим вихідним кодом, що підтримує логічне виведення та запити RDFS. Він пропонує великий набір інструментів для розробників, щоб використовувати можливості RDF і RDF Schema. Це Java-фреймворк, що також має Python-інтерфейс[21].

Окрім того, що він пропонує використовувати нативну базу даних, фрейворк також надає розширений API для створення інших сховищ. Є багато сторонніх сховищ, які можна використовувати за допомогою Sesame Sail API. Прикладами є Bigdata і OWLIM (яка пропонує додаткову підтримку OWL виведення)[4].

1.8.4 GraphDB

GraphDB (колишній OWLIM) - це масштабовний семантичний репозиторій. Він включає в себе сховище триплетів, механізм висновків та обробник SPARQL запитів. GraphDB використовує основу TRREE (Triple Reasoning and Rule Entailment Engine) для виконання RDFS, OWL DLP, OWL Horst виведення та OWL 2 RL[22]. Мови, що підтримуються включають OWL 2 RL, RDFS. GraphDB

пропонує масштабовну підтримку логічного виведення.Доступні як звичайна версія продукту так і хмарна[28].

1.8.5 RDFLib

RDFLib - бібліотека на мові Python для роботи з RDF, потужною мовою для представлення інформації. Бібліотека містить аналізатор та серіалізатор RDF / XML, який відповідає специфікації синтаксису RDF / XML. У бібліотеці також наявні як in-memory, так і постійна графові бази знань.[22].

Щоб вибір програмного забезпечення максимально відповідав необхідним критеріям до продукту з використанням сховища триплетів, необхідно проаналізувати їх за критеріями, що проілюстровані у Таблиці 1:

Таблиця 1.1 - Порівняльний аналіз розглянутих програмних засобів[28]

Назва програмного засобу	**Apache Jena - TDB**	**Dydra**	**Sesame**	**GraphDB**	**RDFLib**
Нативні програмні API	Java	REST API	Java	Java (Jena and RDF4J (Sesame))	Python
Нативна підтримка SPARQL	+	+	+	+	+
Нативна підтримка SPARQL оновлення	+	+	+	+	+
Мова імплементації	Java	Common Lisp, C++	Java	Java	Python
Можливість розгортання у хмарі	+	-(доступ як до сервісу)	+	+	+
Нативна підтримка постійного збереження даних	Сховище триплетів	Графова база даних у хмарі (SaaS)	Сховище триплетів	Сховище триплетів	Berkeley DB
Підтримка роботи з OWL	+	-	-	+	+

Логічне виведення	Pellet, Transitive, RDFS, OWL, OWL Mini, OWL Micro, Generic rule reasoners	-	RDFS reasoning	GraphDB reasoner, OWL DLP, OWL Horst, OWL 2 R reasoning	FuXi reasoner

Звісно, вибір програмного забезпечення залежить від вимог до конкретного сховища триплетів: при необхідності широко використовувати можливості логічного виведення та працювати з OWL-онтологіями як з семантичним доповненням до RDF-даних, найкраще з розглянутих засобів підійде Apache Jena - TDB, оскільки ця система підтримує велику кількість різонерів та здатна працювати з багатьма діалектами OWL[28]. Також для таких цілей, але з меншими можливостями логічного виведення може підійти GraphDB або ж RDFLib (вибір залежить від необхідності розробниками використовувати Java або Python). При необхідності якнайшвидше розгорнути додаток для ефективної роботи зі SPARQL, але без необхідності використовувати семантичні OWL різонери підійде Dydra (за використання REST API) або ж Sesame (за використання Java).

1.9 Висновок

У даному розділі було розглянуто основні поняття для подальшої роботи з семантичним представленням фактів та знань, серед яких:

- Resource Description Framework
- Resource Description Framework Schema
- Web Ontology Language
- Бази знань
- Семантичні різонери
- Мова запитів SPARQL
- Сховища триплетів

База знань (БЗ) - це технологія, яка використовується для зберігання складної структурованої та неструктурованої інформації, яка використовується комп'ютерною системою. Початкове використання цього терміну було пов'язане з експертними системами, які були першими системами, що базуються на знаннях. Сховище триплетів або RDF-сховище є базою даних для зберігання та вилучення триплетів через семантичні запити.

Модель даних RDF схожа на класичні підходи до концептуального моделювання (такі як формування структури сутностей або діаграми класів). Вона заснована на ідеї висловлювати факти про ресурси (зокрема, веб-ресурси) у виразах форми суб'єкт-предикат-об'єкта, відомих як триплети. Суб’єкт позначає ресурс, а предикат позначає риси або аспекти ресурсу і виражає зв'язок між суб'єктом та об'єктом.

SPARQL (рекурсивний акронім від англ. SPARQL Protocol and RDF Query Language) — мова запитів до даних, представлених по моделі RDF, а також протокол для передачі цих запитів і відповідей на них. Використовується для читання та оновлення триплетів у онтологіях.

OWL - це сімейство мов представлення знань для створення авторських онтологій. Онтології є офіційним способом описання таксономій та класифікаційних мережі, істотно визначаючи структуру знань для різних областей.

Семантичний різонер - це програмний компонент для обробки онтологій, що може здійснити певного виду логічні висновки.

Знаючи значення та специфіку понять семантичної мережі, можна набагато детальніше та ефективніше працювати з високо пов’язаними даними, будувати сховища триплетів та інші системи, що використовують технології семантичної павутини.

У даному розділі було також проведено порівняльний аналіз інструментів для роботи з семантичною павутиною, та було обрано фреймворк Apache Jena для подальшої побудови системи “База знань як сервіс”, оскільки ця система підтримує велику кількість різонерів та здатна працювати з багатьма діалектами OWL.

2 ДОСЛІДЖЕННЯ ЕФЕКТИВНОЇ ПОБУДОВИ ВЕБ-СЕРВІСІВ

2.1 Поняття веб-вервісу

На сьогоднішній день доступна величезна кількість платформ для створення веб-продуктів. Використовуються Java, Angular JS, .Net, Node.js та інші. Але таке різноманіття приводить до деяких проблем.

Справа в тому, що додатки різних типів повинні якось взаємодіяти між собою, а оскільки вони розроблені за допомогою різних мов програмування, це створює складність. Виходом з даної ситуації стали веб-сервіси.

Веб-сервіси надають загальну платформу, яка дозволяє різноманітним додаткам обмінюватися даними між собою, незважаючи на різні реалізації.

Веб-сервіс - це набір послуг, який пропонується певним електронним пристрієм іншому шляхом спілкування один з одним через глобальну мережу Інтернет[10]. Веб-сервіси широко застосовують протокол HTTP, що спочатку був призначеним для комунікації між людиною та комп'ютером, а тепер використовується для автоматичного спілкування між машинами, зокрема для передачі форматів файлів, що можуть бути прочитаними машинами, таких як XML та JSON. На практиці веб-служба зазвичай забезпечує об'єктно-орієнтований веб-інтерфейс на сервері баз даних, який використовується, наприклад, іншим веб-сервером або мобільним додатком, який забезпечує користувальницький інтерфейс для кінцевого користувача[10]. Ще однією загальноприйнятою програмою, яка пропонується кінцевому користувачеві, є змішана, де веб-сервер

використовує декілька веб-служб на різних машинах, і збирає вміст у один користувацький інтерфейс.

Веб-сервери можуть також використовувати протокол SOAP замість протоколу HTTP, крім того, веб-сервіси також можуть бути реалізовані за допомогою інших надійних транспортних механізмів, таких як FTP.

Веб-служби характеризуються функціональною сумісністю, розширюваністю та можливістю міжкомп’ютерної (машинно-машинної) взаємодії через мережу [9].

Технологія інтеграції веб-служб дозволяє легко об'єднати прості веб-сервіси, впроваджувати та інтегрувати їх в програмні додатки через глобальну мережу. Можливе об'єднання веб-служб здійснюється незалежно від платформи та мови розробки програми. При використанні веб-сервісів клієнт отримує безпосередньо дані, які можуть бути використані в власних додатках. Веб-сервіс ідентифікується за допомогою URI (Uniform Resource Identifier) і має стандартний інтерфейс. В межах веб-служб взаємодіяють три об'єкта: замовник (замовник послуги), виконавець (провайдер послуг) та реєстр послуг (брокер послуг). В якості реєстру може використовуватися універсальний інтерфейс розпізнавання UDDI (Universal Discovery, Опис та інтеграція), який служить для публікації та виявлення готових веб-сервісів. Веб-сервіси представляють собою реалізацію точно визначених інтерфейсів обміну даними між різними інтернет-додатками, які можуть знаходитися в різних підмережах мережі та працювати на різних програмних і апаратних платформах, написані на різних мовах[10].

2.2 Архітектура REST

Representational State Transfer (REST) - архітектурний стиль, який визначає набір обмежень та властивостей на основі протоколу HTTP. Веб-сервіси, які відповідають архітектурному стилю REST або RESTful веб-сервіси, забезпечують сумісність між комп'ютерними системами в Інтернеті. REST-сумісні веб-сервери дозволяють запитуючим системам отримувати доступ до текстових представлень

веб-ресурсів і керувати ними, використовуючи єдиний та попередньо визначений набір операцій. Інші види веб-сервісів, наприклад веб-служби SOAP, представляють власні довільні набори операцій.

"Веб-ресурси" вперше були визначені в World Wide Web як документи або файли, визначені їх URL-адресами. Однак сьогодні вони мають набагато більш загальне і абстрактне визначення, яке охоплює всі речі або об'єкти, які можуть бути ідентифіковані, названі, адресовані або оброблені будь-яким способом в Інтернеті. У RESTful веб-службі запити, зроблені на URI ресурсу, отримують відповідь, яка може бути в HTML, XML, JSON або іншому форматі. Відповідь може підтвердити, що деякі зміни були внесені в збережений ресурс а також надати гіпертекстові посилання на інші пов'язані ресурси або збірки ресурсів. Коли використовується протокол HTTP, як це найчастіше буває, доступні операції GET, POST, PUT, DELETE та інші попередньо визначені методи CRUD HTTP.

Антиподом REST є підхід, заснований на виклику віддалених процедур (Remote Procedure Call, RPC). Підхід RPC заснований на використанні невеликої кількості мережевих ресурсів з великою кількістю методів і складним протоколом.

2.2.1 Основні ахрітектурні обмеження REST

2.2.1.1 Єдиний інтерфейс (Uniform Interface)

Розглянемо практичну сторони питання. Зрозуміло, що стан нашого додатку - це стан ресурсів, за які він відповідає, і знання того, що можна змінювати його через представлення (ресурсів)[26].

Як змінити стан конкретного ресурсу? Як знайти цей конкретний ресурс? Для цього і придумані URI, щоб ідентифікувати ресурси.

Про URI найкраще думати як про ім'я або псевдонім ресурсу. U - це unified, а не unique. Імен у одного і того ж ресурсу може бути багато. Головне, щоб вони

були зрозумілими для машин. Наприклад, «bank.account.1» і «bank.accs.first» можуть бути цілком реальними ідентифікаторами в певній системі.

URL - це найвідоміший стандарт для ідентифікації ресурсів в інтернеті. URL - це окремий випадок URI, конкретна реалізація[9].

Єдиний інтерфейс визначає інтерфейс між клієнтами та серверами. Це спрощує і відокремлює архітектуру, яка дозволяє кожній частині розвиватися самостійно. Принципи єдиного інтерфейсу:

2.2.1.2 Заснований на ресурсах

Окремі ресурси визначаються в запиті, для чого використовуються URI, як ідентифікатори ресурсів. Самі ресурси концептуально відокремлені від представлень, які повертаються клієнту. Наприклад, сервер не відправляє свою базу даних, а, швидше деякі HTML, XML або JSON, які представляють записи в базі даних, наприклад, в UTF-8, залежно від деталей запиту та реалізації сервера.

2.2.1.3 Маніпуляції над ресурсами через представлення

Коли користувач має представлення про ресурс, у тому числі про пов'язані метадані, він має достатньо інформації для зміни або видалення ресурсу на сервері, якщо він має на це дозволи[26].

2.2.1.4 Повідомлення самі себе документують

Каждое повідомлення містить достатньо інформації для опису того, як його виконати. Наприклад, вызываемый парсер може описиваться за допомогою типового типу Інтернету (так же известний як MIME) Відповіді також явно вказують на їх здатність кешировать.

2.2.1.5 HATEOAS (Hypermedia as the Engine of Application State)

За допомогою HATEOAS клієнт взаємодіє з мережевим додатком, сервер якого забезпечують динамічний доступ через гіпермедіа. REST-клієнту не потрібно заздалегідь знати, як взаємодіяти з додатком або сервером за межею гіпермедіа.

HATEOAS відокремлює клієнта від сервера і дозволяє їм незалежно розвиватися[5].

REST-клієнт звертається до фіксованого URL, а всі наступні дії клієнта стають відомими з ресурсів у відповіді. Типи ресурсів, подання та їх зв'язку стандартизовані. Клієнт проходить по ресурсам, вибираючи посилання або взаємодіючи будь-яким іншим способом, можливим для цього типу ресурсу. Таким чином RESTful-взаємодії працюють через гіпермедіа, а не через заздалегідь зазначений інтерфейс [6].

Наприклад, маємо запит, що повертає ресурс рахунку у вигляді XML-представлення

```
GET /accounts/1000 HTTP/1.1
    Host: somebanksystem.com
    Accept: application/xml
```

Варіант відповіді з сервера, що демонтрує HATEOAS підхід:

```
HTTP/1.1 200 OK
    Content-Type: application/xml
    Content-Length: ...

    <?xml version="1.0"?>
    <account>
      <account_number>1000</account_number>
      <balance currency="usd">1250.00</balance>
      <link rel="deposit" href="https://somebanksystem.com/accounts/1000/deposit" />
      <link rel="withdraw" href="https://somebanksystem.com/accounts/1000/withdraw" />
      <link rel="transfer" href="https://somebanksystem.com/accounts/1000/transfer" />
      <link rel="close" href="https://somebanksystem.com/accounts/1000/close" />
    </account>
```

Відповідь містить посилання на депозит, зняття, перевід і закриття аккаунта

У разі недостатніх коштів на рахунку доступний тільки депозит:

```
HTTP/1.1 200 OK
    Content-Type: application/xml
    Content-Length: ...

    <?xml version="1.0"?>
    <account>
      <account_number>1000</account_number>
      <balance currency="usd">-25.00</balance>
```

```
    <link rel="deposit" href="https://somebanksystem.com/account/12345/deposit" />
</account>
```

Тепер доступне тільки одне посилання: внести більше грошей. Звідси «the Engine of Application State» в назві. Можливі дії розрізняються залежно від стану ресурсу. Клієнту не потрібно знати заздалегідь типи ресурсів і механізми взаємодії з ним, через сервер. Розуміння нових типів ресурсів отримується в режимі реального часу, під час виконання, при отриманні ресурсів від сервера.

2.2.1.6 Відсутність станів(Stateless)

Так як REST це акронім для REpresentational State Transfer, відсутність станів є важливою рисою. Таким чином, це означає, що необхідний стан для обробки запиту міститься в самому запиті, або в рамках URI, параметрах рядка запиту, тіла або заголовках. URI унікально ідентифікує ресурс і тіло містить стан (або зміна стану) цього ресурсу. Потім, після того, як сервер завершить обробку, стан або його частина (і) віддається назад клієнту через заголовки, статус і тіло відповіді[26].

У REST, клієнт повинен включати всю інформацію для сервера для виконання запиту, перевідправляючи стан в разі потреби, якщо цей стан має охоплювати кілька запитів. Відсутність станів забезпечує більшу масштабованість, так як сервер не повинен підтримувати або спілкуватися через стан сеансу. Крім того, балансувальнику навантаження не доведеться враховувати пов'язаність сесії і системи.

Так в чому відмінність між станом і ресурсом? Стан або стан додатку, це те, що сервер може виконати запит для отримання даних необхідних для поточної сесії або запиту. Ресурсний стан, або ресурс - це дані, які визначають представлення ресурсу, наприклад, представлення даних, що зберігаються в базі даних.

2.2.1.7 Кешування відповіді (Cacheable)

Клієнт може кешувати відповіді. Таким чином, відповіді явно або неявно визначають себе як такі, які можна кешувати або ж ні, для запобігання

повторного використання клієнтами застарілих або некоректних даних у відповідь на подальші запити. Добре спроектоване кешування частково або повністю усуває деякі клієнт-серверні взаємодії, сприяючи подальшій масштабованості та продуктивності[26].

2.2.1.8 Клієнт-сервер (Client-Server)

Єдиний інтерфейс відокремлює клієнтів від серверів. Поділ інтерфейсів означає, що, наприклад, клієнти не пов'язані зі зберіганням даних, яке залишається всередині кожного сервера, так що мобільність коду клієнта поліпшується[26]. Сервери не пов'язані з інтерфейсом користувача або станом, так що сервери можуть бути простішими і масштабованішими. Сервери та клієнти можуть бути замінні і розроблятися незалежно, поки інтерфейс не змінюється.

2.2.1.9 Багаторівнева система (Layered System)

Зазвичай клієнти не можуть сказати - вони підключені безпосередньо до сервера або спілкуються через посередника. Проміжний сервер може поліпшити масштабованість системи, забезпечуючи балансування навантаження і надаючи загальний кеш. Шари також можуть відповідати за політику безпеки. Тобто REST передбачає можливість кластерної / розподіленої роботи серверної частини, але при цьому для клієнта сервіс повинен залишатися цілісним і не вимагати додаткових даних / дій від клієнта.

2.2.1.10 Код на вимогу (Code on Demand - опціонально)

Сервери можуть тимчасово розширювати функціонал клієнта, передаючи йому логіку, яку він може виконувати. Наприклад, це можуть бути скомпільовані Java-аплети або клієнтські скрипти на Javascript

Дотримуючись цих обмежень, і, таким чином, дотримуючись RESTful архітектури, ми дозволяємо розподіленій системі будь-якого типу мати такі властивості як[26]:

- продуктивність
- розширюваність

- простота
- оновлюваність
- зрозумілість
- портативність
- надійність

2.3 Архітектура SOAP

SOAP (англ. *Simple Object Access Protocol*) — протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

Специфікація SOAP може бути широко визначена, що складається з наступних 3 концептуальних компонентів: концепції протоколів, концепцій інкапсуляції та концепцій мережі.

2.3.1 Концепції протоколу

SOAP має певний набір правил, що формалізують та регулюють правила форматування та обробки інформації, яка обмінюється між відправником SOAP та приймачем SOAP.

SOAP-вузли це фізичні / логічні машини з процесорами, які використовуються для передачі / перенаправлення, отримання та обробки SOAP-повідомлень. Вони аналогічні мережевим вузлам.

На шляху SOAP-повідомлення всі вузли беруть на себе певну роль. Роль вузла визначає дію, яку вузол виконує для отриманого повідомлення. Наприклад, роль "none" означає, що жоден вузол ніяким чином не оброблятиме заголовок SOAP і просто не передаватиме повідомлення по його шляху.

SOAP-повідомлення повинні працювати разом з іншими протоколами, що підлягають передачі через мережу. Наприклад, повідомлення SOAP може

використовувати TCP як протокол нижчого рівня для передачі повідомлень. Ці прив'язки визначаються у схемі зв'язування протоколу SOAP [13].

SOAP забезпечує тільки обмін повідомленнями. Тим не менш, він може бути розширений, щоб додати такі функції, як надійність, безпека тощо. Є правила, яких слід дотримуватися при додаванні функцій до SOAP-системи.

Модуль SOAP - це набір специфікацій щодо семантики заголовка SOAP для опису будь-яких нових функцій, що розповсюджуються на SOAP. Модуль повинен реалізувати 0 або більше функцій. SOAP вимагає, щоб модулі дотримувалися встановлених правил.

2.3.2 Концепції інкапсуляції даних

SOAP-повідомлення представляє інформацію, що обмінюється між 2 SOAP вузлами.

Конверт SOAP це сусідній елемент XML-повідомлення, який ідентифікує його як SOAP-повідомлення.

Блок заголовка SOAP може містити більше одного з цих блоків, кожен з яких є дискретним обчислювальним блоком у заголовку. Загалом інформація про роль SOAP використовується для визначення вузлів на шляху.

Заголовок SOAP - це збірка з одного або декількох блоків заголовків, орієнтованих на кожного SOAP-приймача.

SOAP тіло містить тіло повідомлення, призначеного для приймача SOAP. Інтерпретація та обробка тіла SOAP визначається блоками заголовків.

Несправність SOAP використовується у випадку, якщо вузол SOAP не обробляє SOAP-повідомлення; тоді додається інформація про помилку до елементу помилки SOAP. Цей елемент міститься в тілі SOAP як дочірній елемент.

2.3.3 Концепції відправника та отримувача повідомлення

Відправник SOAP - це вузол, який передає SOAP-повідомлення.

SOAP-приймач - це вузол, що приймає повідомлення SOAP. (Може бути посередником або цільовим вузлом.)

Шлях повідомлення SOAP - це шлях, що складається з усіх вузлів, до яких проходить SOAP-повідомлення для досягнення цільового вузла.

Початковий відправник SOAP - це вузол, з якого було створено SOAP-повідомлення для передачі. Це корінь шляху повідомлення SOAP.

Посередник SOAP: всі вузли між вихідним джерелом SOAP та призначенням SOAP. Кожен з них обробляє блоки заголовків SOAP, орієнтовані на нього, і здійснює передачу SOAP-повідомлення до кінцевого приймача SOAP.

Остаточний приймач SOAP - це приймач призначення SOAP-повідомлення. Цей вузол відповідає за обробку тіла повідомлення та будь-яких заголовних блоків, націлених на нього.

2.3.4 Транспортні методі SOAP

І SMTP, і HTTP є валідними протоколами прикладного рівня, які використовуються як транспортний шар для SOAP, але HTTP отримав ширше схвалення, оскільки він добре працює з сучасною інфраструктурою Інтернету; зокрема, HTTP добре працює з мережевими брандмауерами. SOAP також може бути використаний через протокол HTTPS (який є тим самим протоколом, що і HTTP на рівні додатків, але використовує зашифрований транспортний протокол під) як з простою, як і взаємною автентифікацією; це захищений метод WS-I для забезпечення безпеки веб-сервісу, як зазначено у базовому профілі WS-I 1.1.

Це є основною перевагою перед іншими розподіленими протоколами, такими як GIOP / IIOP або DCOM, які, як правило, відфільтровуються міжмережевими екранами. SOAP над AMQP - це ще одна можливість, яку підтримують деякі реалізації.

2.4 Порівняльний аналіз SOAP і REST

Як було опубліковано у попередньому дослідженні “The knowledge base as a service architectural requirements research” [30], важливим є вибір архітектурного стилю для проектування бази знань як сервісу. Порівнявши найбільш часто використовувані архітектури SOAP (Simple Object Access Protocol) та REST (Representational State Transfer), було визначено, що REST краще підходить для вирішення таких задач. Серед його переваг над SOAP можна виділити наступні:

- SOAP - це ціле сімейство протоколів і стандартів, звідки напряму випливає, що це більш важкий і складний варіант з точки зору машинного оброблення. Тому REST працює швидше.
- SOAP використовує HTTP як транспортний протокол, в той час як REST базується на ньому. Це означає, що всі існуючі напрацювання на базі протоколу HTTP, такі як кешування на рівні сервера, масштабування, продовжують працювати також в REST-архітектурі, а для SOAP необхідно шукати інші засоби.
- Відповідь REST може бути представлена в різних форматах, а SOAP прив'язаний до XML.
- SOAP працює з операціями, а REST - з ресурсами.
- Всі запити у REST можна поділити на 4 типи у відповідності з CRUD, причому кожен тип відповідає HTTP методу - Post, Get, Put, Delete
- Всі ресурси у REST мають унікальний ідентифікатор (URI)
- Всі операції клієнта з сервером у REST не використовують збереження стану, тобто сервер не повинен зберігати взагалі ніякої інформації про клієнта.

Таблиця 2.1 Порівняння архітектур REST та SOAP[Розробка автора]

№ п/п	SOAP	REST
1.	SOAP - це протокол	REST - це архітектурний стиль
2.	SOAP розшифровується як Simple Object Access Protocol.	REST розшифровується як REpresentational State Transfer.
3.	SOAP не може використовувати REST, оскільки це протокол.	REST може використовувати веб-сервери SOAP, оскільки це більше концепція ніж стандарт і може використовувати будь-який протокол, як HTTP, SOAP.
4.	SOAP працює з операціями	REST використовує URI для виявлення бізнес-логіки, працює з ресурсами
5.	SOAP вимагає більшої пропускної спроможності та ресурсу, ніж REST.	REST вимагає меншої пропускної здатності та ресурсу, ніж SOAP.
6.	SOAP допускає лише формат даних XML.	REST дозволяє використовувати різні формати даних, такі як звичайний текст, HTML, XML, JSON тощо.

2.5 Висновки:

Отже, веб-сервіси представляють собою реалізацію точно визначених інтерфейсів обміну даними між різними Інтернет-додатками, які можуть знаходитися в різних підмережах мережі та працювати на різних програмних і апаратних платформах, написані на різних мовах. У даному розділі було розглянуто цілі створення веб-сервісів та проблеми, які вони вирішують, можливі підходи до їх реалізації. Було проаналізовано архітектурний стиль REST та сформульовано його основні ахрітектурні обмеження, серед яких:

- єдиний інтерфейс (Uniform Interface)
- відсутність станів (Stateless)
- кешування відповіді (Cacheable)
- клієнт-серверна архітектура (Client-Server)
- багаторівнева система (Layered System)

- код на вимогу (Code on Demand - опціонально)

Було також розглянуто протокол SOAP, його особливості та можливості. Було наведено порівняльний аналіз протоколу SOAP та архітектурного стилю REST. Для подальшого проектування додатку “База знань як сервіс” було обрано підхід REST, тому що відповідь REST може бути представлена в різних форматах, а SOAP прив'язаний до XML, SOAP працює з операціями, а REST - з ресурсами, всі ресурси у REST мають унікальний ідентифікатор (URI), а операції клієнта з сервером у REST не використовують збереження стану, тобто сервер не повинен зберігати взагалі ніякої інформації про клієнта.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ З ДОСТУПОМ ДО СХОВИЩА ТРИПЛЕТІВ

3.1 Формулювання вимог до програмного продукту

Прикладні інформаційні системи можуть взаємодіяти з RDF-сховищем за допомогою спеціального API або структурованих SPARQL-запитів. SPARQL - це стандартна мова запитів до RDF-даних. Як було досліджено у “The knowledge base as a service architectural requirements research” [30], для найбільш ефективної роботи з базою знань за допомогою сервісу можна виділити такі вимоги[30]:

- Можливість виконання SPARQL-запитів.
- Наявність REST-API з реалізацією доступу до ресурсів, які найбільш часто будуть потрібні користувачам (тут важливою є постановка вимог до системи кінцевими користувачами для визначення таких ресурсів).
- Можливість розгортання системи у хмарі (для забезпечення кращої масштабовності та універсальності).
- Кросплатформність (дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення).

Визначення вимог до бази знань як сервісу може значно скоротити час на вибір програмного забезпечення, бібліотек та фреймворків для її реалізації. Це також допоможе уникнути проблем при необхідності масштабувати чи розгортати систему у різноманітних (у тому числі хмарних) середовищах. Отже, дане дослідження є підґрунтям для подальшої реалізації бази знань як сервісу[30].

3.2 Аналіз інструментів для реалізації сховища триплетів

Приймаючи до уваги аналіз програмних засобів, за допомогою яких можна створити сховище триплетів, при необхідності широко використовувати можливості логічного виведення та працювати з OWL-онтологіями як з

семантичним доповненням до RDF-даних, найкраще з розглянутих засобів підійде Apache Jena - TDB, оскільки ця система підтримує велику кількість різонерів та здатна працювати з багатьма діалектами OWL.

Тим паче, даний фреймворк розробений на мові Java, а використавши саме цю мову для побудови системи, можна забезпечити одну з вимог до неї - кросплатформність.

3.2.1 Мова програмування Java

Мова програмування Java спочатку розроблялася компанією Sun Microsystems, яку ініціював Джеймс Гослінг і випустив у 1995 році як основний компонент платформи Java (Java 1.0 (J2SE)) Sun Microsystems.Останній випуск стандартної версії Java - це Java SE 10. З розвитком Java та її широкої популярності було створено кілька конфігурацій для різних типів платформ. Наприклад: J2EE для корпоративних додатків, J2ME для мобільних додатків. Джава має такі характеристики:

1. Об’єктно-орієнтовна . У Java все є об'єктом. Java можна легко розширити, оскільки вона заснована на моделі Object.
2. Кросплатформна. На відміну від багатьох інших мов програмування, включаючи C і C ++, Java код компілюється в поатформно-незалежний байтовий код. Цей байтовий код поширюється через Інтернет та інтерпретується віртуальною машиною (JVM) на будь-якій платформі, на якій вона запускається.
3. Проста. Java розроблена так, щоб легко вивчатись розробниками. Якщо ви розумієте основну концепцію OOP Java, її легко освоїти.
4. Безпечна. За допомогою функціоналу безпеки Java це дає змогу розробляти системи, що не містять вірусів. Методи аутентифікації засновані на шифруванні з відкритим ключем.

5. Архітектурно-нейтральна . Java-компілятор породжує формат об'єктно-архітектурно-нейтрального об'єкта, що робить скомпільований код виконуваним на багатьох процесорах за наявності JRE.
6. Портативна. Будучи архітектурно-нейтральною і не маючи залежних від виконання функцій специфікацій, вона є портативною.
7. Надійна . Java докладає зусиль для усунення ситуацій, пов'язаних із помилками, підкреслюючи головним чином перевірку помилок часу компіляції та перевірку помилок часу виконання.
8. Багатопотокова. За допомогою багатопотокового функціоналу Java можна писати програми, які можуть виконувати багато завдань одночасно. Цей функціонал дозволяє розробникам створювати інтерактивні додатки, які можуть працювати без проблем[12].
9. Високопродуктивна. Завдяки використанню компіляторів Just-In-Time, Java забезпечує високу продуктивність.

Джеймс Гослінг ініціював проект зі створення мови Java у червні 1991 року. Мова спочатку називалася "Oak" в честь дуба, що ріс біля офісу Гослінга, пізніше була перейменована в "Java" зі списку випадкових слів.

Sun випустив першу публічну реалізацію Java 1.0 у 1995 році. Слоганом був Write Once, Run Anywhere(Написавши раз, запускай будь-де)

13 листопада 2006 року Sun випустив більшу частину Java як вільне та відкрите програмне забезпечення за умовами Загальної публічної ліцензії GNU (GPL).

8 травня 2007 року Sun завершив процес, зробивши все основне забезпечення Java вільним та відкритим джерелом, крім невеликої частини коду, на яке Sun не моє авторського права[12].

3.2.2 Apache Jena TDB

Jena - це Java API для семантичних веб-додатків. Ключовим RDF-пакетом розробника додатків є org.apache.jena.rdf.model. API визначено з точки зору інтерфейсів, щоб код програми міг працювати з різними реалізаціями без змін. Цей пакет містить інтерфейси для представлення моделей, ресурсів, властивостей, літералів, тверджень та інших ключових понять RDF, а також ModelFactory для створення моделей. Таким чином, код програми залишається незалежним від реалізації, найкраще, якщо він використовує інтерфейси, де це можливо, а не конкретні реалізації класу[2].

Apache Jena TDB є компонентом Jena для зберігання та запиту RDF. Він підтримує весь спектр API Jena. TDB може використовуватися як високопродуктиве RDF-сховище на одній машині[2].

Набір даних із підтримкою TDB зберігається в єдиному каталозі у файловій системі. Набір даних має такі складові:

- Таблиця вузлів
- Індекси триплетів та кводів(quadres)
- Таблиця префіксів

Таблиця вузлів зберігає представлення термінів RDF. Вона забезпечує два відображення від вузла до його Id та від Id до вузла. Це також називають словником.

Мапінг від вузла до його Id(NodeId) використовується під час завантаження даних і при перетворенні констант у запитах з представлення їх у вигляді Jena Node до специфічних внутрішніх ідентифікаторів TDB[2].

Відображення Id в вузол використовується для перетворення результатів запиту, виражених як NodeIds TDB, у представлення Jena Node, а також під час обробки запитів при застосуванні фільтрів, якщо для перевірки потрібне все представлення вузла (наприклад, регулярний вираз).

Реалізація таблиць вузлів зазвичай забезпечує великий кеш - відображення NodeId в Node значною мірою використовується для обробки запитів, однак той самий NodeId може з'явитися в багатьох результатах запиту.

NodeId займають по 8 байт. Відображення NodeId до NodeId базується на хеші Node (128-бітний хеш MD5)[2].

Зберігання за замовчуванням для таблиці вузлів - це послідовний файл доступу для відображення NodeId до Node і дерево B + для узгодження з NodeId.

Індекси триплетів та кводів

Кводи використовуються для іменованих графів, триплети для графа за замовчуванням. Таблиця триплетів - це три індекси, кожен з яких містить всю інформацію про триплет.

За замовчуванням зберігається кожен індекс.

У таблиці префіксів використовується таблиця вузлів та індекс для GPU (Назва графа(Graph)-> Префікс(Prefix)-> URI). Зазвичай він невеликий. Він не бере участі в обробці запитів. Він надає підтримку компоненті Jena PrefixMapings, що використовується переважно для презентації та для серіалізації трійок у RDF / XML або Turtle[2].

TDB B + Дерева

Багато стійких структур даних у TDB використовують спеціальну реалізацію “threaded” дерев B +. Реалізація TDB забезпечує тільки фіксовану довжину ключа(key) та фіксовану довжину значення(value). Значення у індексах не використовуються[2].

“Threaded” означає, що тривале сканування індексів відбувається без необхідності перетинати гілки дерева.

3.3 Аналіз засобів для створення веб-сервісу

3.3.1 Фреймворк Spring

Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники, метою яких є створення систем на платформі Java. Через широку функціональність важко визначити найважливіші структурні елементи, з яких він складається.

Spring, ймовірно, найбільш відомий як джерело розширень (особливостей), необхідних для ефективної розробки складних бізнес-додатків, що не мають важких програмних моделей, які історично були домінуючими в галузі. Ще одна його перевага в тому, що він вніс раніше невикористані функціональні можливості в сьогоднішні основні методи розробки, навіть поза платформою Java[32].

Цей фреймворк пропонує послідовну модель і робить її прикладною для більшості типів додатків, які вже створені на базі платформи Java. Вважається, що Spring реалізує модель розробки, засновану на кращих стандартах промисловості, і робить її доступною у багатьох областях Java.

Центральна частина Spring - це контейнер Inversion of Control, який надає засоби конфігурування та керування об'єктами Java за допомогою рефлексії. Контейнер відповідає за управління життєвим циклом об'єкта: його створення, виклик методів ініціалізації та конфігурації; зв’язки з іншими об’єктами шляхом їх зв'язування між собою[32].

Об'єкти, створені контейнером, також називаються керованими об'єктами (бінами). Зазвичай конфігурація контейнера здійснюється шляхом завантаження XML-файлів, що містять визначення біля і надають інформацію, необхідну для створення файлів.

Об'єкти можуть бути отримані одним з двох способів[32]:

- Пошук залежностей - шаблон проектування, в якому виклик об'єкта запитує в об'єкті-контейнера екземпляр об'єкту з певним ім'ям або певним типом.
- Впровадження залежності - шаблон проектування, в якому контейнер передає екземпляри об'єктів за їх іменем іншими об'єктами за допомогою конструктора, властивостей або фабричного методу.

Spring Boot дозволяє легко створювати автономні, продуктивні Spring додатки, які можна "просто запустити". Більшість застосунків Spring Boot потребують дуже малої конфігурації Spring.

Особливості Spring Boot[32]:

- Надає швидке Створення окремих програм Spring
- Має вбудований веб-сервер Tomcat, Jetty або Undertow безпосередньо (не потрібно розгортати файли WAR)
- Наявний початковий POM файл для спрощення налаштування Maven
- Готовий функціонал для надання метрик, перевірки стану та наявна зовнішня конфігурація.
- Не вимагає конфігурації XML

3.4 Опис архітектурних рішень програмної реалізації

Було розроблено багатошаровий веб-сервіс з доступом до сховища триплетів. Для кращої гнучкості та можливості розширювати та удосконалювати додаток у майбутньому, було вирішено дотримуватись принципів предметно-орієнтованого проектування.

Предметно-орієнтоване проектування (Domain-driven design, DDD) - це підхід до моделювання складного об'єктно-орієнтованого програмного забезпечення[9]. Переваги DDD полягають в наступному:

- Широке розуміння предметної області за допомогою представлення її у вигляді доменних класів

- Створення програмних моделей, які відображують глибоке розуміння предметної області.

Даний підхід і термін, який його визначає, винайшов Ерік Еванс і продемонстрував його основні принципи у свої книзі з однойменною назвою.

Основні шари для побудови додатку згідно з DDD:

- Доменні об’єкти, що відображають основні предметні сутності додатку.
- Сервіси слугують для делегації бізнес-логіки у їх функціонал.
- Репозиторії потрібні для отримання об'єктів предметної області. Якщо є певний інтерфейс, якому повинна відповідати реалізація, можна підмінювати місце збереження об'єктів[9].

Згідно з предметно-орієнтованим підходом до проектування, у додатку є шар сервісів, репозиторіїв, доменних об’єктів та контролерів для HTTP доступу до ресурсів додатку. Також наявні два DTO (Data Transfer Objects) класи для трансляції JSON запитів у Java об’єкти(OntologyURIDTO, SPARQLQueryDTO).

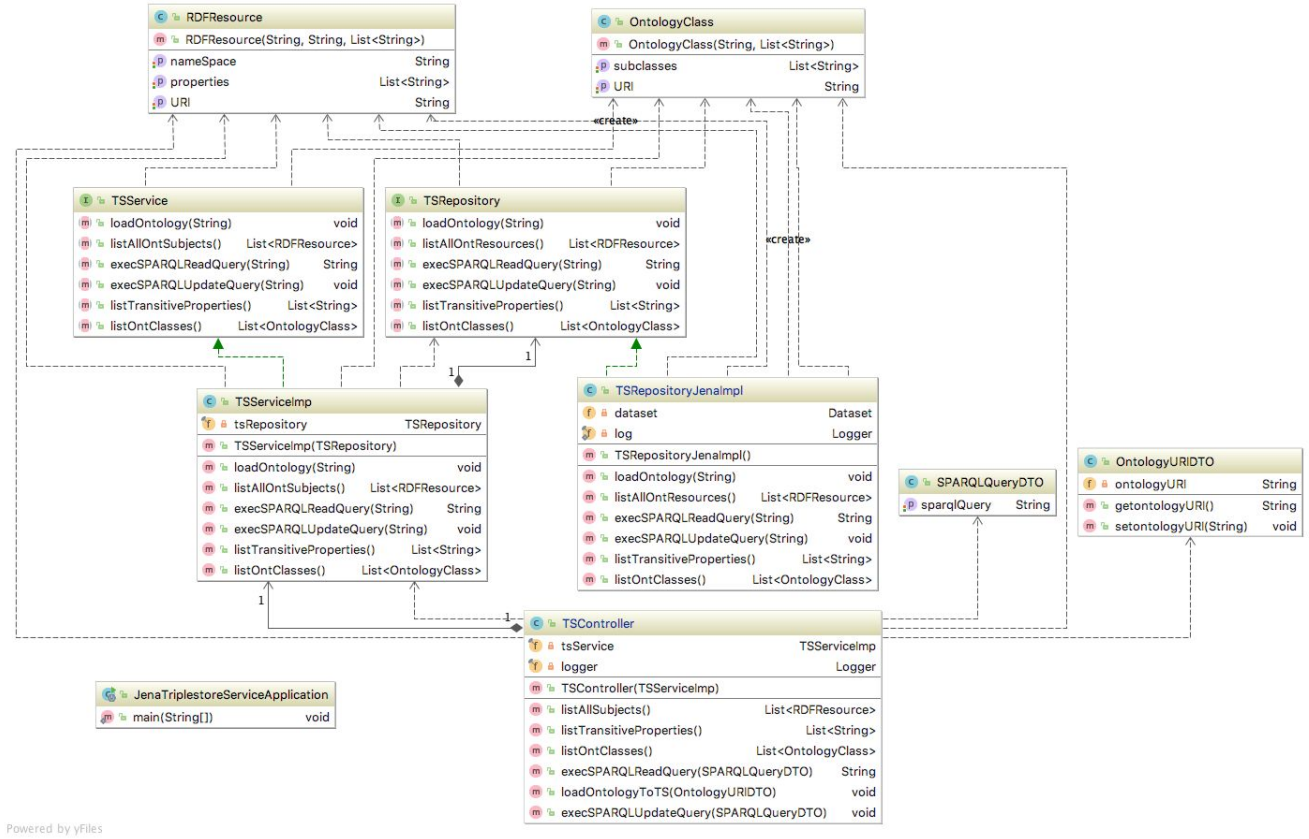

Рисунок 3.1 - Діаграма класів додатку

Для завантаження необхідних модулів був використаний Maven.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.restdocs</groupId>
        <artifactId>spring-restdocs-mockmvc</artifactId>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.jena/apache-jena-libs -->
    <dependency>
        <groupId>org.apache.jena</groupId>
        <artifactId>apache-jena-libs</artifactId>
        <version>3.6.0</version>
        <type>pom</type>
    </dependency>
```

Рисунок 3.2 - Основні Maven залежності додатку у файлі pom.xml

3.4.1 Опис доменних об’єктів

Для реалізації необхідного функціоналу було розроблено два доменних об’єкти:

OntologyClass - клас для опису об’єктів, що містять інформацію про класи в онтологіях, а саме URI класу онтології та список його підкласів.

RDFResource - клас для опису об’єктів, що містять інформацію про суб’єкти RDF, а саме URI суб’єкта, список його властивостей та назву простору імен, до якого він належить.

```
public class RDFResource {
    private String URI;
    private List<String> properties;
    private String nameSpace;

    public RDFResource(String URI, String nameSpace, List <String> properties){
        this.URI = URI;
        this.nameSpace = nameSpace;
        this.properties = properties;
    }
```

Рисунок 3.3 - Поля та публічний конструктор класу RDFResource

```
public class OntologyClass {
    private String URI;
    private List<String> subclasses;

    public OntologyClass(String URI, List<String> subclasses){
        this.URI = URI;
        this.subclasses = subclasses;
    }
```

Рисунок 3.4 - Поля та публічний конструктор класу OntologyClass

3.4.2 Опис шару репозиторіїв

Було представлено інтерфейс TSRepository, у якому наявні такі методи:

- void loadModel(String path);
- List<RDFResource> listAllOntResources();
- String execSPARQLReadQuery(String sparqlQuery) ;
- void execSPARQLUpdateQuery(String sparqlQuery) ;
- List<String> listTransitiveProperties() ;
- List<OntologyClass> listOntClasses() ;

Метод loadOntology приймає URI онтології як вхідний параметр типу String, яку користувач бажає завантажити у сховище триплетів. В результаті дана онтологія завантажується у базу знань.

Метод listAllOntResources зазначає, що будь-який репозиторій, який надалі реалізовуватиме цей інтерфейс, повинен надавати функціонал переліку всіх об’єктів онтологій у сховищі триплетів у вигляді списку доменних об’єктів RDFResource.

Метод execSPARQLReadQuery приймає на вхід параметр, у якому наявний код SPARQL-запиту на читання, та надає результат його виконання у форматі String(JSON).

Метод execSPARQLUpdateQuery приймає на вхід параметр, у якому наявний код SPARQL-запиту на оновлення, та виконує зміну у структурі сховища триплетів згідно до запиту.

Метод listTransitiveProperties заявляє, що будь-який репозиторій, який надалі реалізовуватиме цей інтерфейс, повинен надавати можливість переліку всіх транзитивних зв’язків у сховищі триплетів у вигляді списку.

Метод listOntClasses заявляє, що будь-який репозиторій, який надалі реалізовуватиме цей інтерфейс, повинен надавати можливість переліку всіх класів онтологій у сховищі триплетів у вигляді списку доменних об’єктів OntologyClass.

Реалізовано даний інтерфейс було у Java-класі TSRepositoryJenaImpl. З назви зрозуміло, що методи даного класу будуть реалізовані за допомогою фреймворку Jena.

Розгляньмо реалізацію інтерфейсу TSRepositoryJenaImpl детальніше. Анотація @Repository потрібна для того, щоб Spring міг ініціалізувати об’єкт нашого репозиторію. Дана анотація не лише повідомляє про те, що TSRepositoryJenaImpl є біном контексту Spring, а й інформує розробників про те, що дана сутність працюватиме з якимось сховищем даних та надаватиме певний доступ до них. Така конструкція підвищує якісь програмного коду та його підтримку.

У класі наявні два поля та публічний конструктор, що ініціалізує сховище триплетів у каталозі “triplestore” та виводить інформацію про успішність операції в лог.:

```
@Repository
public class TSRepositoryJenaImpl implements TSRepository{
    private Dataset dataset;
    private static final Logger log = LoggerFactory.getLogger( TSRepositoryJenaImpl.class );

    public TSRepositoryJenaImpl(){
        dataset = TDBFactory.createDataset( dir: "triplestore");
        log.info("Dataset was successfully created");
    }
```

Рисунок 3.5 - Поля та публічний конструктор класу TSRepositoryJenaImpl

Поле dataset типу Dataset надає змогу створити каталог у файловій системі, який буде використовуватись для зберігання триплетів. Також за допомогою цього об'єкту можна створювати об'єкти, що реалізують інтерфейс Model з фреймворку Jena для подальшої взаємодії зі сховищем триплетів.

```
public void loadOntology(String path)
{
    Model model = null;

    dataset.begin( ReadWrite.WRITE );
    try
    {
        model = dataset.getDefaultModel();
        FileManager.get().readModel( model, path );
        dataset.commit();
    }
    finally
    {
        log.info("Loading of " + path + " ontology is complete ");
        dataset.end();
    }
}
```

Рисунок 3.6 - Реалізація методу loadOntology

У метод loadOntology передається шлях URI до онтології, яку користувач бажає завантажити в сховище триплетів. Далі починається транзакція на запис за допомогою команди dataset.begin(ReadWrite.WRITE). Далі отримуємо об'єкт типу Model за допомого наступної команди: model = dataset.getDefaultModel(). Потім зчитується потрібна онтологія за допомогою команди FileManager.get().readModel(model, path) та виконується комін транзакції за допомогою виклику dataset.commit();

```
public List<RDFResource> listAllOntResources(){
    dataset.begin( ReadWrite.READ );
    List<RDFResource> allOntResources = new ArrayList<>();
    try {
        Model model = dataset.getDefaultModel();
        ResIterator it = model.listSubjects();
        while (it.hasNext()) {
            Resource node = it.next();
                List<String> properties = new ArrayList<String>();
                StmtIterator stmtIt = node.listProperties();
                while (stmtIt.hasNext()) {
                    Statement st = stmtIt.next();
                    properties.add(st.getObject().toString());
                }
                RDFResource nextRDFresource = new RDFResource(node.asResource().getURI(),
                                                              node.asResource().getNameSpace(),
                                                              properties);
                node.asResource().getNameSpace();
                allOntResources.add(nextRDFresource);
        }
        dataset.commit() ;
    } finally {
    dataset.end() ;
    }
    return allOntResources;
}
```

Рисунок 3.7 - Реалізація методу listAllOntResources

Оскільки у методі listAllOntResources потрібно виконати вибірку усіх суб’єктів зі сховища з їхніми властивостями, URI, та назвами просторів імен, результат він повертає у вигляді списку доменних об’єктів RDFResource. Спочатку починаємо транзакцію на читання за допомогою команди dataset.begin(ReadWrite.READ). Отримуємо модель для роботи зі сховищем за допомогою виклику Model model = dataset.getDefaultModel(). За допомогою цієї моделі зчитуємо всі суб’єкти за допомогою команди model.listSubjects(). Далі у циклі для отриманого ітератору зчитуємо для кожного суб’єкту список властивостей properties за допомогою команди node.listProperties(). Коли маємо всі необхідні дані для окремого суб’єкта, викликаємо конструктор RDFResource(node.asResource().getURI(),node.asResource().getNameSpace(), properties), таким чином створюючи доменний об’єкт типу RDFResource з URI, назвою простору імен та властивостями.

В результаті роботи методу, всі отримані суб’єкти будуть записані в список allOntResources для подальшої роботи з ними.

```
public List<OntologyClass> listOntClasses() {
    dataset.begin(ReadWrite.READ);
    List <OntologyClass> ontClasses = new ArrayList<OntologyClass>();
    try {
        Model model = dataset.getDefaultModel();
        OntModel ontModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, model);
        ExtendedIterator<OntClass> iterator  = ontModel.listClasses();
        while (iterator.hasNext()) {

            OntClass ontClass = iterator.next();
            ExtendedIterator<OntClass> listSubClasses= ontClass.listSubClasses();
            List <String> classInstances = new ArrayList<>();

            while (listSubClasses.hasNext()){
                OntClass subclass = listSubClasses.next();
                classInstances.add(subclass.getURI());
            }

            OntologyClass domainOntologyClass = new OntologyClass(ontClass.getURI(), classInstances);
            ontClasses.add(domainOntologyClass);
        }
        dataset.commit() ;
    } finally {
        dataset.end() ;
    }
    return ontClasses;
}
```

Рисунок 3.8 - Реалізація методу listOntClasses

Клас TSRepositoryJenaImpl також реалізовує метод listOntClasses, який надає результат у вигляді списку доменних об’єктів OntologyClass.

У даному методі робота зі сховищем уже проводиться за допомогою моделі OntModel, яка надає можливість роботи з сутностями OWL, такими як класи, підкласи, транзитивні відношення і тд. Щоб створити таку модель, необхідно передати у конструктор стандартну модель і діалект OWL, з яким буде працювати модель: ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, model). За допомогою виклику методу listClasses() отримуємо список класів сховища триплетів, далі для кожного з отриманих класів викликаємо метож listSubClasses для отримання URI їх підкласів. Коли маємо всі необхідні дані для окремого класу, викликаємо конструктор OntologyClass(ontClass.getURI(), classInstances), таким чином створюючи доменний об’єкт типу OntologyClass з URI та підкласами.

В результаті роботи методу, всі отримані суб’єкти будуть записані в список ontClasses для подальшої роботи з ними.

```
public List<String> listTransitiveProperties() {
    dataset.begin(ReadWrite.READ);
    List transitiveProps = new ArrayList<String>();
    try {
        Model model = dataset.getDefaultModel();
        OntModel ontModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, model);
        ExtendedIterator<TransitiveProperty> iterator  = ontModel.listTransitiveProperties();

        while (iterator.hasNext()) {
            TransitiveProperty transProp = iterator.next();
            transitiveProps.add(transProp.getURI());

        }
        dataset.commit() ;
    } finally {
        dataset.end() ;
    }
    return transitiveProps;
}
```

Рисунок 3.9 - Реалізація методу listTransitiveProperties

У методі listTransitiveProperties знову ж таки працюємо з моделлю OntModel, яка надає можливість роботи з сутностями OWL, такими як класи, підкласи, транзитивні відношення і тд. За допомогою виклику методу ontModel.listTransitiveProperties отримаємо URI всіх транзитивних властивостей у сховищі триплетів.

```
public String execSPARQLReadQuery(String sparqlQuery) {
    dataset.begin(ReadWrite.READ);
    QueryExecution qExec = QueryExecutionFactory.create(sparqlQuery, dataset);
    ResultSet rs = qExec.execSelect();
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    ResultSetFormatter.outputAsJSON(outputStream, rs);
    String jsonRS = new String(outputStream.toByteArray());
    dataset.commit();
    return jsonRS;
}
```

Рисунок 3.10 - Реалізація методу execSPARQLReadQuery

Для отримання результату SPARQL-запиту на читання, використовується об’єкт QueryExecution qExec = QueryExecutionFactory.create(sparqlQuery, dataset), якому передається на вхід dataset та сам текст запиту. Результати отримуються у вигляді об’єкту ResultSet, який далі форматуємо у JSON за допомогою виклику методу ResultSetFormatter.outputAsJSON(outputStream, rs) та повертаємо у об’єкті типу Sring.

```
public void execSPARQLUpdateQuery(String sparqlQuery) {
    dataset.begin(ReadWrite.WRITE);
    try {

        GraphStore graphStore = GraphStoreFactory.create(dataset);
        UpdateRequest request = UpdateFactory.create(sparqlQuery);
        UpdateProcessor proc = UpdateExecutionFactory.create(request, graphStore);
        proc.execute();

        dataset.commit();
    } finally {
        dataset.end();
    }
}
```

Рисунок 3.11 - Реалізація методу execSPARQLUpdateQuery

У методі execSPARQLUpdateQuery відкриваємо транзакцію на запис, після чого за допомогою об'єкту UpdateProcessor виконуємо запит на оновлення. Далі виконується коміт транзакції і закінчення роботи з dataset.

3.4.3 Опис шару сервісів

Було розроблено інтерфейс TSService з наступними методами:

- void loadOntology(String URIpath)
- List<RDFResource> listAllOntSubjects()
- String execSPARQLReadQuery(String sparqlQuery)
- void execSPARQLUpdateQuery(String sparqlQuery)
- List<String> listTransitiveProperties()
- List<OntologyClass> listOntClasses()

По суті методи сервісу викликатимуть одноіменні методи з репозиторію, який буде впроваджено у реалізований клас TSServiceImpl:

```
@Service
public class TSServiceImp implements TSService {

    private final TSRepository tsRepository;

    @Autowired
    public TSServiceImp(TSRepository tsRepository) {
        this.tsRepository = tsRepository;
    }
```

Рисунок 3.12 - Впровадження залежності у клас TSServiceImpl

3.4.4 Опис шару контролерів

Було розроблено контролер TSController з впровадженою залежністю сервісу в ньому:

```
@Controller
public class TSController {

    private final TSServiceImp tsService;

    private final Logger logger = LoggerFactory.getLogger(TSController.class);

    @Autowired
    public TSController(TSServiceImp tsService) {
        this.tsService = tsService;
    }
```

Рисунок 3.13 - Впровадження залежності через конструктор у клас TSController

У даному контролері було реалізовано три методи для обробки HTTP GET запитів та три методи для обробки HTTP POST запитів від користувачів.

Реалізацію обробки HTTP GET запитів можна продемонструвати за допомогою одного з трьох реалізованих методів listAllSubjects.

```
@GetMapping(value = "/getdata/get-all-subjects")
@ResponseBody
public List<RDFResource>  listAllSubjects() {
    return tsService.listAllOntSubjects();

}
```

Рисунок 3.14 - Реалізація методу listAllSubjects

За допомогою анотації @GetMapping вказуємо, що метод буде обробляти запити типу HTTP GET зі значенням “/getdata/get-all-subjects”. Анотація @ResponseBody забезпечує автоматичне перетворення отриманого списку об’єктів RDFResource в формат JSON. Аналогічно працюють і два інші методи контролера, такі як listTransitiveProperties та listOntClasses.

Три інші методи класу оброблятимуть запити типу HTTP POST. Розглянемо детальніше такі методи за допомогою демонстрації роботи методу execSPARQLReadQuery.

За допомогою анотації @RequestMapping вказуємо, що метод буде обробляти запити типу HTTP POST зі значенням “/postdata/update-sparql”. Анотація @ResponseStatus вказує, що за успішного завершення методу, буде відправлено HTTP статус 200, що означає успішне виконання операції. Анотація @RequestBody автоматично перетворить вхідний JSON на об’єкт типу SPARQLQueryDTO, що надасть можливість працювати далі з даними запиту у потрібному вигляді.

```
@RequestMapping(method = RequestMethod.POST,
        value = "postdata/update-sparql")
@ResponseStatus(HttpStatus.OK)
public void  execSPARQLUpdateQuery (@RequestBody SPARQLQueryDTO sparqlDTO) {
        tsService.execSPARQLUpdateQuery(sparqlDTO.getSparqlQuery());
}
```

Рисунок 3.15 - Реалізація методу execSPARQLUpdateQuery

Два інші методи loadOntologyToTS та execSPARQLReadQuery реалізовані за допомогою аналогічних конструкцій фреймворку Spring.

3.5 Хмарне розгортання додатку

Хероку - хмарна платформа (PaaS), яка підтримує кілька мов програмування, що використовується як модель розгортання веб-додатків. Хероку, одна з перших хмарних платформ, була розроблена з червня 2007 року, коли вона підтримувала лише мову програмування Ruby, але зараз підтримує Java, Node.js, Scala, Clojure, Python, PHP і Go. З цієї причини, Heroku дозволяє розробнику розробляти, запускати і масштабувати програми аналогічним способом на всіх мовах. Heroku був придбаний Salesforce.com у 2010 році за 212 мільйонів доларів[17].

Щоб розгорнути Spring Boot додаток у даній системі, необхідно зробити такі кроки:

1. Створити безкоштовний обліковий запис Heroku. Потім завантажити та встановити Heroku CLI.

2. Після встановлення, необхідно використати команду heroku login з терміналу для входу за допомогою електронної адреси та пароля, які ви використовували при створенні облікового запису Heroku:

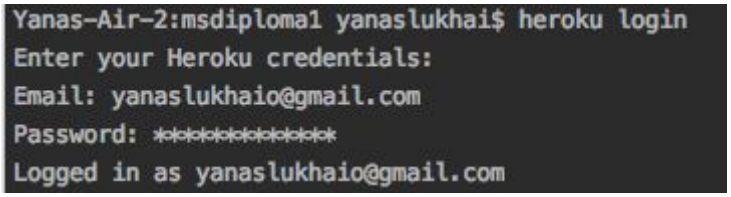

Рисунок 3.16 - Процес авторизації у обліковий запис Heroku

3. Перш ніж розгорнути додаток до програми Heroku, потрібно створити сховище Git для програми та додати до нього весь код. Код додатку уже був збережений у системі контролю версій Git:

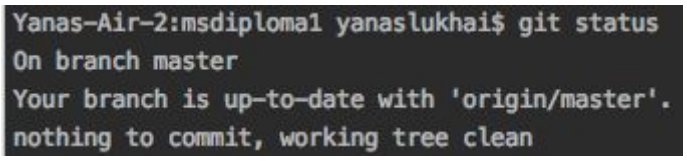

Рисунок 3.17 - Наявність коду додатку у системі Git

4. Для того, щоб розгорнути на Heroku, спочатку потрібно буде створити новий додаток Heroku за допомогою команди heroku create[17].

Ця команда також створює віддалений репозиторій, який називається heroku, у місцевому реєстрі git. Heroku генерує випадкове ім'я для вашого додатка. Ви можете пізніше перейменувати його за допомогою програм heroku.

5. Тепер можна розгорнути код:

```
Yanas-Air-2:msdiploma1 yanaslukhai$ git push heroku master
Counting objects: 46, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (37/37), done.
Writing objects: 100% (46/46), 62.05 KiB | 0 bytes/s, done.
Total 46 (delta 19), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Java app detected
remote: -----> Installing JDK 1.8... done
remote: -----> Executing: ./mvnw -DskipTests clean dependency:list install
remote:        /tmp/build_b75da416397e6dd81003bd7a22749c6a
remote:        [INFO] Scanning for projects...
remote:        [INFO]
remote:        [INFO] ----------------< com.diploma:jena-triplestore-service >----------------
remote:        [INFO] Building jena-triplestore-service 0.0.1-SNAPSHOT
remote:        [INFO] --------------------------------[ jar ]---------------------------------
remote:        [INFO]
```

Рисунок 3.18 - Процес доставки коду у Heroku

```
remote:        [INFO] ------------------------------------------------------------------------
remote:        [INFO] BUILD SUCCESS
remote:        [INFO] ------------------------------------------------------------------------
remote:        [INFO] Total time: 6.837 s
remote:        [INFO] Finished at: 2018-05-09T12:25:15Z
remote:        [INFO] ------------------------------------------------------------------------
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> web
remote:
remote: -----> Compressing...
remote:        Done: 75.1M
remote: -----> Launching...
remote:        Released v5
remote:        https://glacial-plateau-25490.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/glacial-plateau-25490.git
   289d0b8..8963781  master -> master
```

Рисунок 3.19 - Успішна збірка додатку у Heroku

Пересвідчитись у успішності розгортання останньої версії додатку можна у розділі Last Activity:

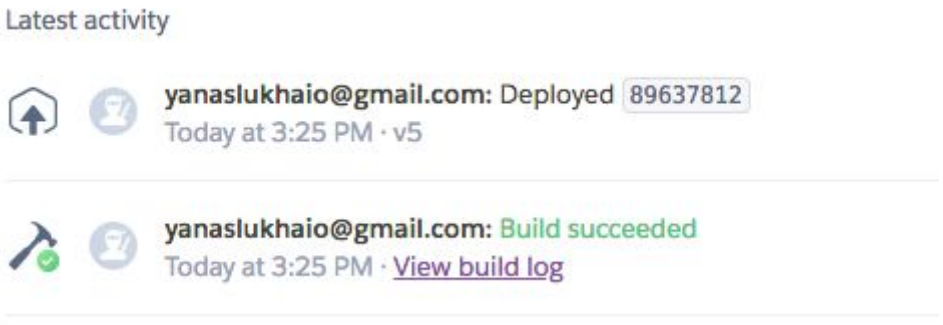

Рисунок 3.20 - Інформація про успішну збірку останньої версії коду додатку на Heroku

3.5 Тестування програмного продукту

Для тестування створеного додатку будуть використані онтології з BioPortal - сховища біомедичних онтологій. BioPortal надає доступ до найбільш часто використовуваних біомедичних онтологій. Завантажмо у сховище триплетів дві онтології:

1. https://bitbucket.org/uamsdbmi/dron/raw/master/dron-upper.owl
2. https://bitbucket.org/uamsdbmi/dron/raw/6bcc56a003c6c4db6ffbcbca04e10d2712fadfd8/dron-hand.owl

Усі запити до веб-сервісу будуть продемонстровані за допомогою додатку Postman.

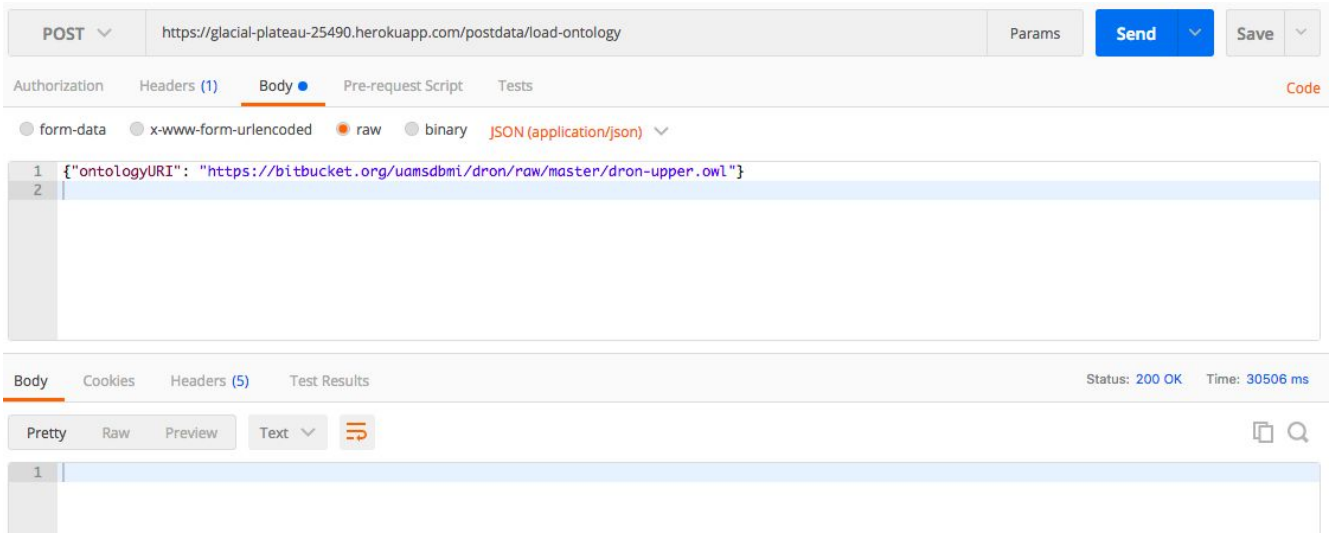

Рисунок 3.21 - Тестування завантаження онтології №1 у сховище триплетів

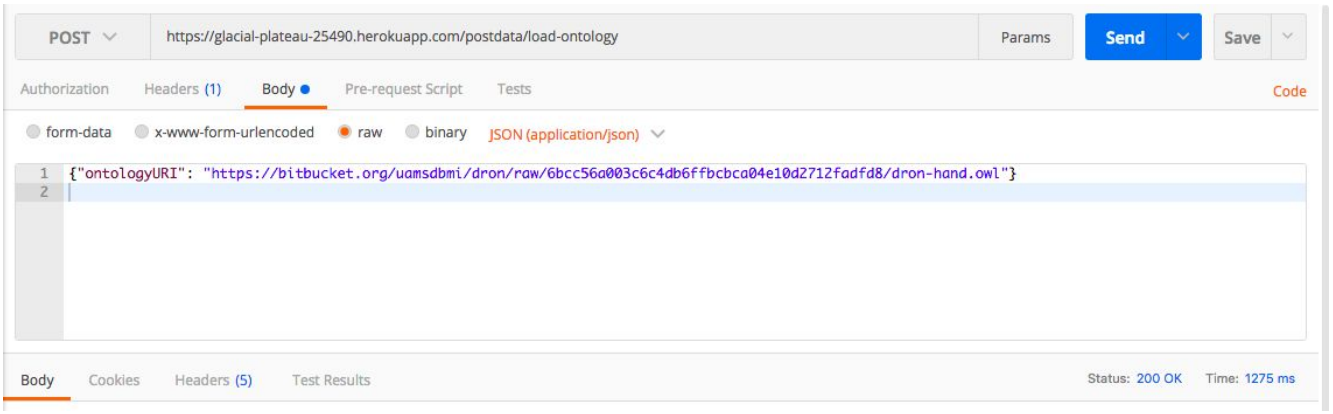

Рисунок 3.22 - Тестування завантаження онтології №2 у сховище триплетів

Для тестування наявності даних у сховищі можемо скорстатись GET методом https://glacial-plateau-25490.herokuapp.com/getdata/get-all-subjects. Метод працює і повертає відповідь у вигляді JSON списку з URI суб'єктів, їх властивостями та назвою простору імен. Фрагмент відповіді якої зображено на рис. 3.23

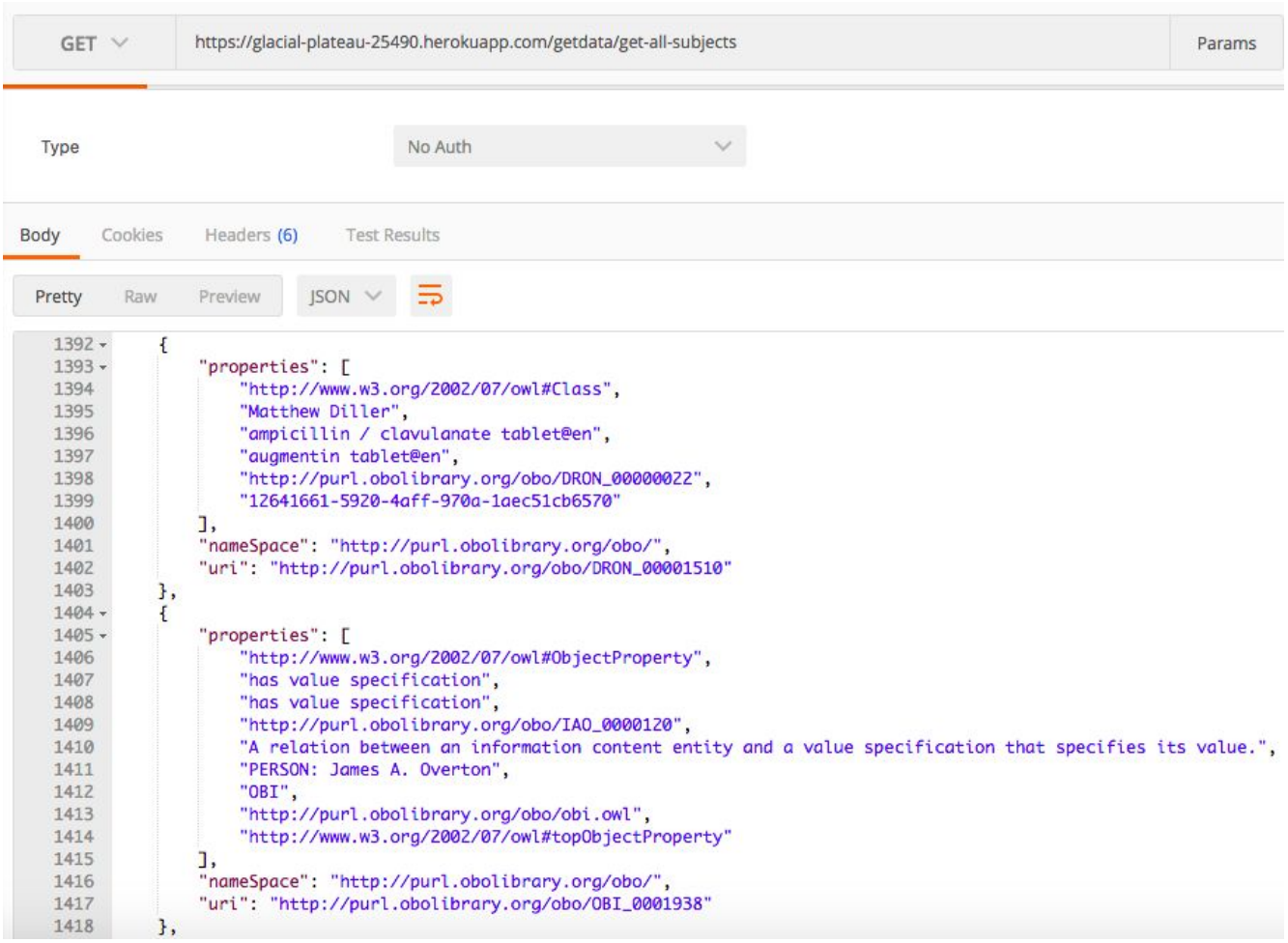

Рисунок 3.23 - Тестування GET методу зі значенням “getdata/get-all-subjects”

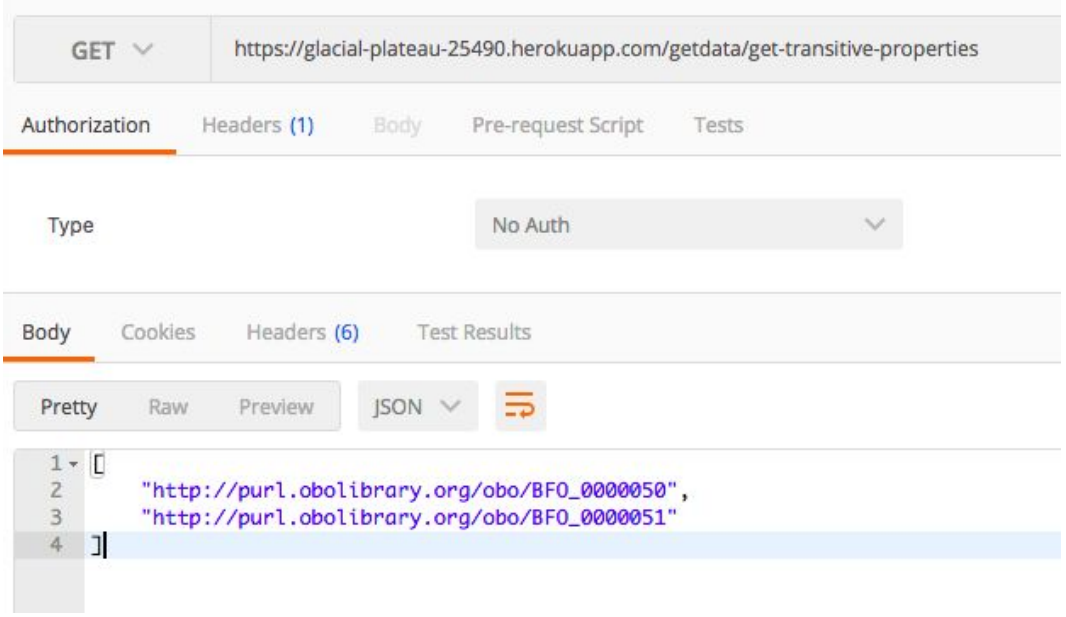

Рисунок 3.24 - Тестування GET методу зі значенням “/getdata/get-transitive-properties”

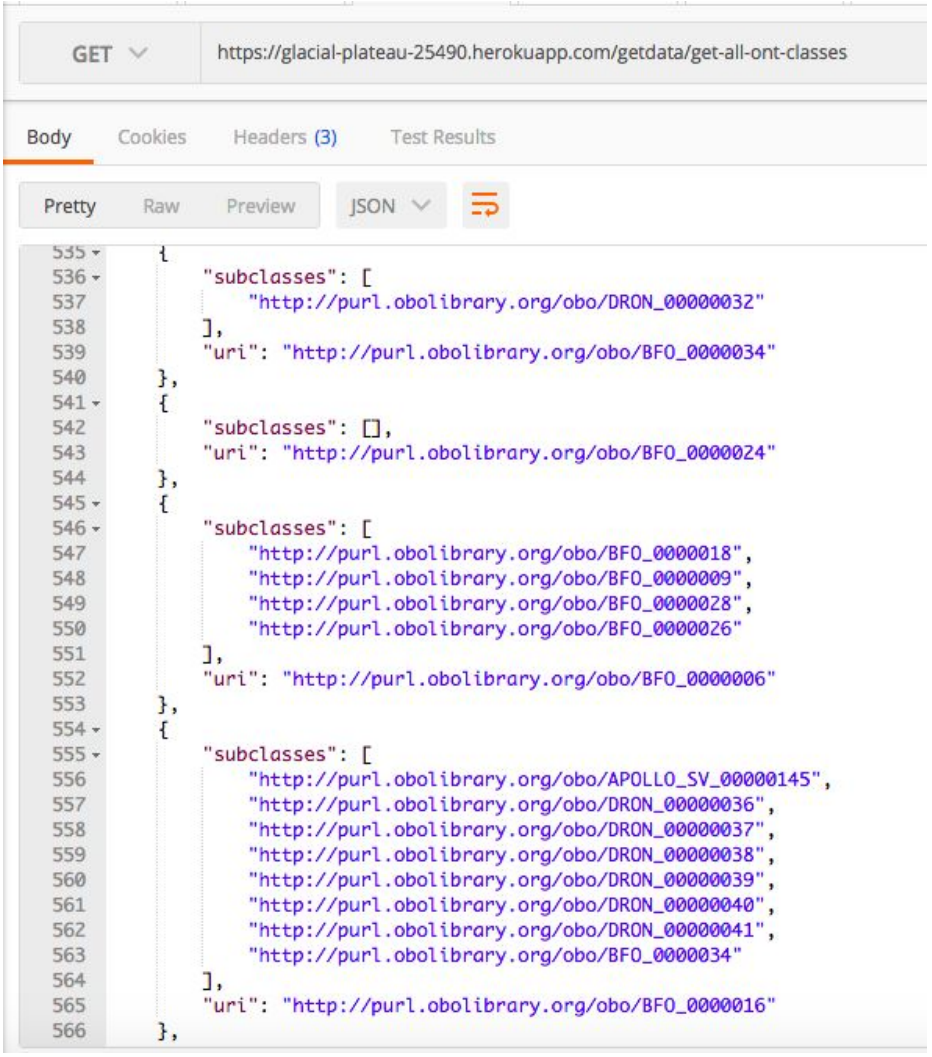

Рисунок 3.25 - Тестування GET методу зі значенням “/getdata/get-all-ont-classes”

Метод GET методу зі значенням “/getdata/get-all-ont-classes” працює і повертає відповідь у вигляді JSON списку з URI класів онтологій, та URI їх підкласів. Фрагмент відповіді зображено на рис. 3.25.

Далі протестуємо роботу SPARQL-ендпойнту:

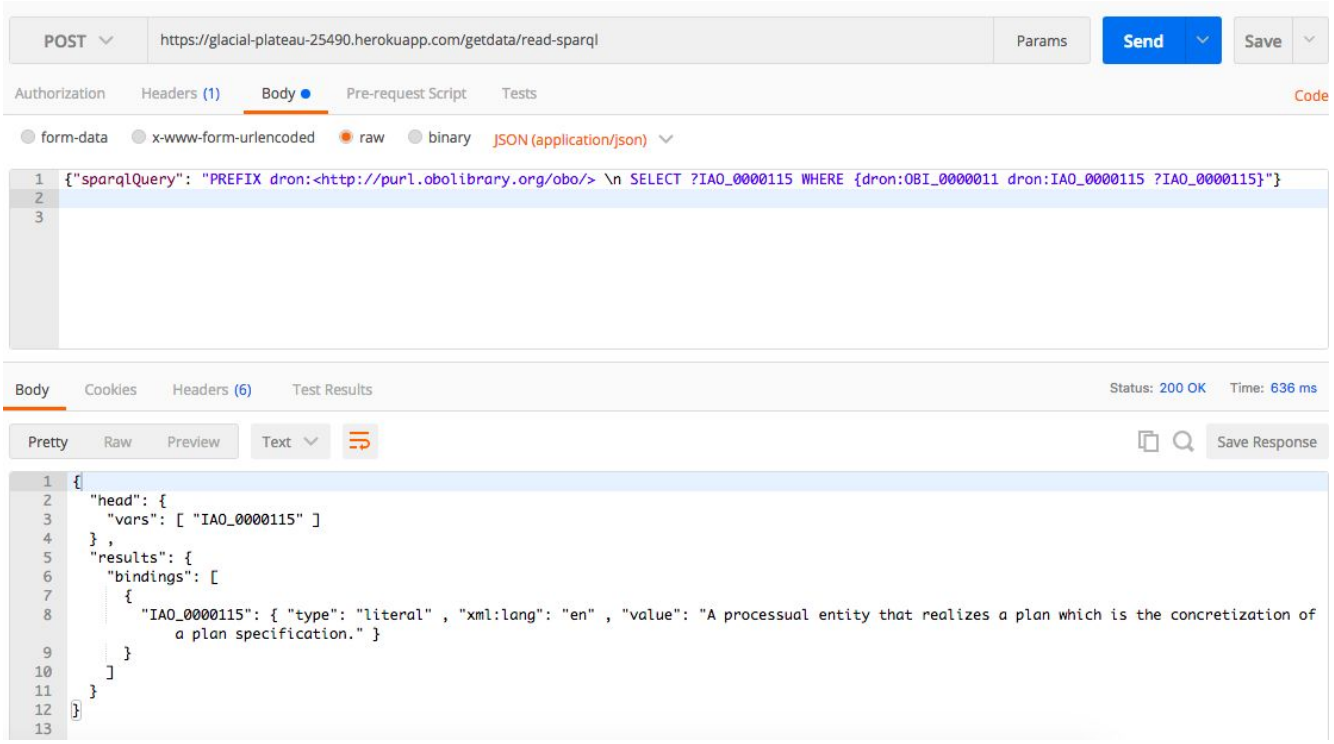

Рисунок 3.26 - Тестування SPARQL-ендпойнту на операцію читання

Відправляється наступний запит:

```
PREFIX dron:<http://purl.obolibrary.org/obo/>
SELECT ?IAO_0000115 WHERE
{dron:OBI_0000011 dron:IAO_0000115 ?IAO_0000115}
```

Отже, ми хочемо отримати значення властивості http://purl.obolibrary.org/obo/IAO_0000115 для суб'єкту http://purl.obolibrary.org/obo/OBI_0000011. Отримане значення співпадає з очікуваним:

```
    <!-- http://purl.obolibrary.org/obo/OBI_0000011 -->

    <owl:Class rdf:about="http://purl.obolibrary.org/obo/OBI_0000011">
        <rdfs:subClassOf rdf:resource="http://purl.obolibrary.org/obo/BFO_0000015"/>
        <dron:IAO_0000111 xml:lang="en">planned process</dron:IAO_0000111>
        <dron:IAO_0000112 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Injecting mice with a vaccine in order to test its
efficacy</dron:IAO_0000112>
        <dron:IAO_0000114 rdf:resource="http://purl.obolibrary.org/obo/IAO_0000122"/>
        <dron:IAO_0000115 xml:lang="en">A processual entity that realizes a plan which is the concretization of a plan specification.
</dron:IAO_0000115>
```

Рисунок 3.27 - Фрагмент онтології зі значенням властивості http://purl.obolibrary.org/obo/IAO_0000115.

Наступним методом для тестування є метод POST зі значенням “postdata/update-sparql”. Відправляємо наступний SPARQL-запит на вставку нового триплеу:

INSERT {<http://purl.obolibrary.org/obo/T> <http://purl.obolibrary.org/obo/IAO_0000115> <http://purl.obolibrary.org/obo/test>} WHERE{}"

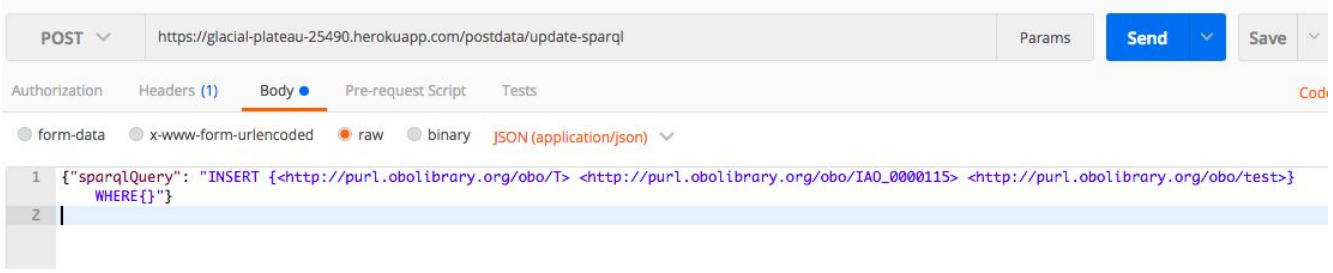

Рисунок 3.28 - Виконання вставки нового тестового триплету

Виконаємо наступний SPARQL-запит для перевірки наявності нового триплету в базі знань:

SELECT ?IAO_0000115 WHERE {<http://purl.obolibrary.org/obo/T> <http://purl.obolibrary.org/obo/IAO_0000115> ?IAO_0000115}

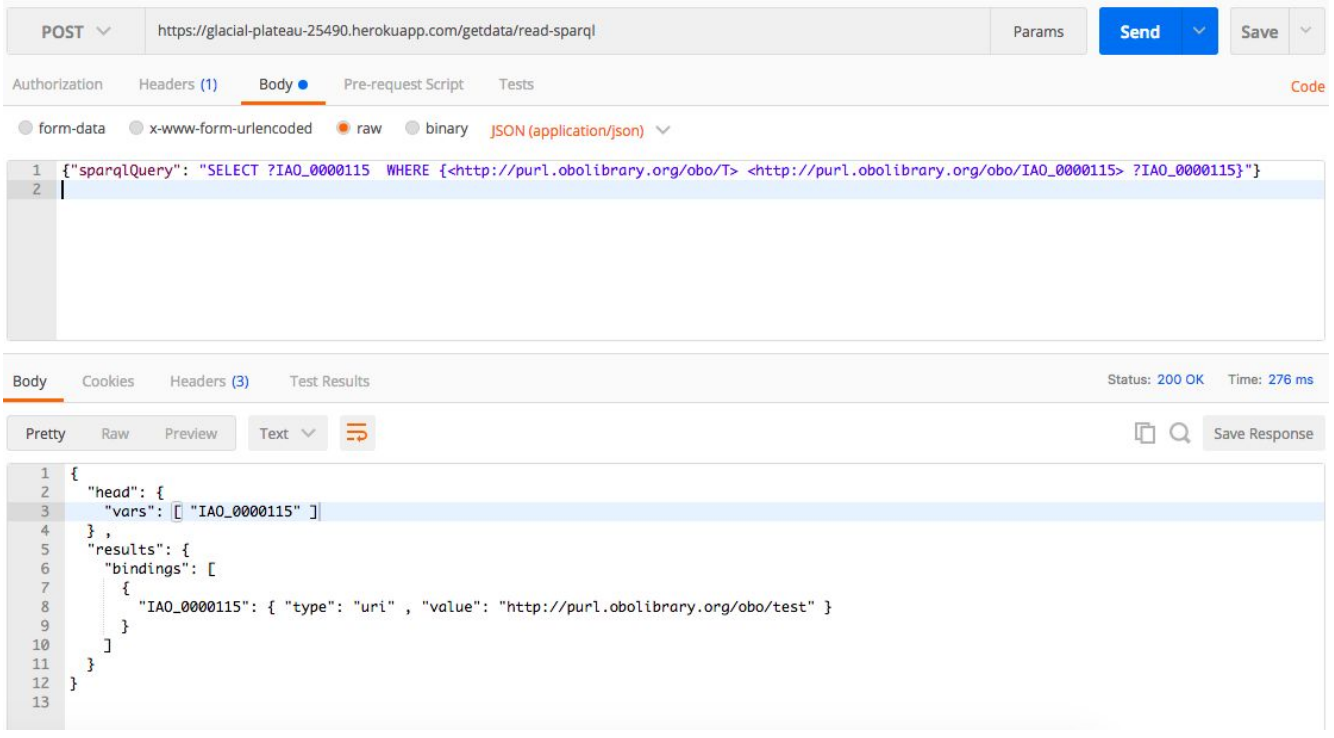

Рисунок 3.29 - Перевірка наявності вставленого триплету у базі знань.

Бачимо, що вставлений триплет з’явився у базі знань.

3.6 Висновок

У даному розділі було розглянуто архітектурні рішення для реалізації додатку “База знань як сервіс”. Було обрано предметно-орієнтований підхід (Domain Driven Development) для проектування системи, адже такий спосіб структурування шарів програми надає можливість простого розширення існуючого додатку та ізолює бізнес-логіку від роботи з репозиторіями даних.

В якості фреймворку для розробки RESTful API було обрано Spring Boot, оскільки він спрощує розробку Spring додатків з нуля та зменшує кількість необхідних конфігурацій для початку роботи над системою.

Було розроблено систему ”База знань як сервіс” у вигляді RESTful API з такими можливостями:

- Перелік класів онтологій у сховищі та їх підкласів
- Перелік RDF суб’єктів та їх властивостей
- Перелік транзитивних відношень у сховищі триплетів
- SPARQL-ендпойнт для оновлення триплетів
- SPARQL-ендпойнт для читання триплетів
- Завантаження у сховище цілих онтологій за їх URI

Під час розробки додатку було використано такі шаблони проектування як Data Transfer Object, Dependency Injection, SOLID, MVC. Було надано діаграму класів для кращої візуалізації архітектури програми.

Додаток було розгорнуто у хмарному середовищі Heroku. Було надано детальний опис процесу такого розгортання.

Робота додатку була проілюстрованою у середовищі для розробників API Postman у окремому підрозділі.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ “БАЗА ЗНАНЬ ЯК СЕРВІС”

4.2 Опис ідеї стартап-проекту

Розділ має на меті проведення маркетингового аналізу стартап проекту “База знань як сервіс” задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Опис стартап-проекту наведено у Таблиці 4.1.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Створення та розгортання бази знань у хмарі з наданням користувачеві доступу читання та запису через REST api. Надання SPARQL-ендпойнту для читання та оновлення даних у сховищі триплетів, можливості завантажувати цілі онтології, знаходити транзитивні відношення, класи та об’єкти онтологій.	1. Використання для виконання аналізу даних, що зберігаються у вигляді онтологій.	REST API надає можливість використовувати систему як сервіс, що значно спрощує доступ до сховища. Знаходження транзитивних відношень, класів та об’єктів онтологій може спростити подальший аналіз даних.
	2. Використання сховища триплетів у інших додатках як спосіб збереження сильно пов’язаних доменних даних.	Можливість виконання SPARQL запитів дає можливість гнучкого доступу до даних та їх оновлення. Сервіс легко інтегрувати у інші системи завдяки REST API

Отже, проект “База знань як сервіс” може бути використаним як інструментом для деякого аналізу даних, так і прошарком постійного збереження сильно пов’язаних доменних даних у інфраструктурі інших додатків завдяки можливості використання даної системи як сервісу через REST API та реалізованому SPARQL-ендпойнту.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характерис-тики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтра- льна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент 1*	*Конкурент2*	*Конкурент 3*			
1.	Форма виконання	Веб-сервіс	Програма	Веб-додаток	Програма			+
2.	Собівартість	Низька	Низька	Висока	Висока			+
3.	Кросплатформність	Так	Ні	Так	Так			+
4.	Наявність SPARQL-ендпойнту для зчитування даних	Так	Так	Так	Так		+	
5.	Наявність SPARQL-ендпойнту для оновлення даних	Так	Ні	Ні	Так			+
6.	Застосування логічного виведення	Так	Ні	Так	Так			+
7.	Горизонтальне масштабування	Ні	Так	Ні	Так	+		

Сильними сторонами проекту є форма виконання у вигляді веб-сервісу, низька собівартість, кросплатформність, Наявність SPARQL-ендпойнту для оновлення даних, застосування логічного виведення. Слабкою стороною є

відсутність можливості горизонтального масштабування, нейтральною - наявність SPARQL-ендпойнту для зчитування даних. Отож, система є конкурентноспроможною.

4.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Створення сховища триплетів	Apache Jena TDB	Наявна	Безкоштовна, доступна
		Dydra	Наявна	Частково безкоштовна
2.	Створення REST API для доступу до бази знань	Spring Boot, Maven	Наявні	Безкоштовна, доступна
		Amazon API Gateway, AWS Lambda	Наявна	Платні
3.	Хмарне розгортання додатку	Heroku	Наявна	Безкоштовна, доступна
		AWS EC2, S3	Наявна	Платні

Обрані технології реалізації ідеї проекту: Apache Jena TDB через повну безкоштовність фреймворку та наявність докладної документації, наявність досвіду роботи розробників з даною технологією; Spring Boot, Maven через простоту використання, безкоштовність та можливість розгортання додатків на основі таких технологій у хмарі; Heroku для розгортання у хмарі через безкоштовність.

4.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	5
2.	Загальний обсяг продаж, грн/ум.од	8000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Отже, було проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*

1.	Необхідне програмне забезпечення(REST API) для доступу до бази знань як до сервісу	Потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує обробки сильно пов’язаних даних	Цільова група займається дослідженнями або має обробляти дані у вигляді онтологій та RDF-графів	Рішення повинне бути придатним до інтеграції в інші більш складні системи, мати SPARQL-endpoint, бути здатним надавати перелік класів, об’єктів і транзитивних зв’язків у сховищі триплетів бути розгорнутим у хмарі

Згідно проведеної характеристики потенційних клієнтів стартап-проекту випливає, що на ринку є затребуваним програмне забезпечення(REST API) для доступу до бази знань як до сервісу і потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує обробки сильно пов’язаних даних.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. № 6-7). Фактори в таблиці подавати в порядку зменшення значущості.

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок великої компанії	1. Вихід з ринку 2. Запропонувати великій компанії поглинути себе 3. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1. Передбачити можливість додавання нового функціоналу до створюваного ПЗ
3.	Зміна тарифів	Необхідність оплати	1. Пошук іншого

	провайдера хмарного розгортання на платні	послуг провайдера хмари	безкоштовного провайдера 2. Пошук інвестицій для оплати існуючого провайдера
4.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів
5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Отже, було проаналізовано фактори загроз ринкового впровадження проекту, серед яких: конкуренція, уповільнення росту ринку, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні та надходження на ринок альтернативних продуктів. Було також запропоновано можливі реакції компанії.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів	Наявність попиту на інструменти для обробки даних у вигляді онтологій та RDF-графів	Змога запропонувати продукт більшій кількості потенційних користувачів
2.	Поява нових різонерів	Надання нового функціоналу для надання результатів роботи нового логічного виведення	Розробка нового функціоналу у вигляді нового HTTP запиту для надання користувачам результатів логічного виведення
3.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшій кількості потенційних користувачів
4.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі
5.	Розширення	Поява нового	Розробка нового

	асортименту можливих послуг	функціоналу, що привабить нових користувачів	функціоналу, що є потребою певної групи користувачів

У Таблиці 4.7 наведено фактори можливостей ринкового впровадження проекту, серед яких: стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів, поява нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг; було також запропоновано можливі реакції компанії.

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - досконала	Існує 3 фірми-конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Одна з компаній – з іншої країни, дві – з України	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю - не марочна	Бренди відсутні	-

У Таблиці 4.8 наведено ступеневий аналіз конкуренції на ринку, де було визначено особливості конкурентного середовища та їх вплив а діяльність підприємства. Однією з найбільш важливих дій компанії для досягнення конкурентоспроможності є необхідність створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (Табл. 9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складов і аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
	Навести перелік прямих конкурентів	*Визначити бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачі в*	*Фактори загроз з боку замінників*
Висновки:	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-додатку	Так, можливості для входу на ринок є, бо наше рішення має SPARQL-ендпойнт для оновлення даних та можливість логічного виведення	Постачальники відсутні.	Важливим для користувача є швидкість роботи ПЗ	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

Було здійснено аналіз конкуренції в галузі за М. Портером, в результаті чого було визначено, що існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-додатку, але можливості для входу на ринок є, бо наше рішення має SPARQL-ендпойнт для оновлення даних та можливість логічного виведення.

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспромож- ності у п. 3.6. 3.6) На основі аналізу конкуренції, проведеного в п. 3.5 (табл. 9), а та- кож із урахуванням характеристик ідеї проекту (табл. 2), вимог споживачів до товару (табл. 5) та факторів маркетингового середовища (табл. № 6-7) визначається та

обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 10

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Наявність SPARQL-endpoint для зчитування та оновлення даних	Дозволяє користувачам здійснювати гнучкий доступ до бази знань
2.	Можливість завантажувати цілі онтології	Висока швидкість заповнення бази знань
3.	Наявність REST API	Дозволяє інтегрувати сервіс у складні системи завдяки універсальному API
4.	Хмарне розгортання	Дозволяє звертатись до бази знань як до сервісу
5.	Горизонтальне масштабування	Можливість гнучкого масштабування за допомогою додавання апаратних компонентів

У Таблиці 7 наведено обґрунтування факторів конкурентоспроможності, серед яких: наявність SPARQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API та хмарне розгортання. Було також наведено обґрунтування цих факторів.

За визначеними факторами конкурентоспроможності (табл. 10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 11).

У наступній таблиці наведено проведення аналізу сильних та слабких сторін стартап-проекту, факторами конкурентоспроможності виступили такі: наявність SPARQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API, хмарне розгортання, горизонтальне масштабування.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			-3	-2	-1	0	1	2	3
1.	Наявність SPARQL-endpoint для зчитування та оновлення даних	20				+			
2.	Можливість завантажувати цілі онтології	20			+				
3.	Наявність REST API	15			+				
4.	Хмарне розгортання	15		+					
5.	Горизонтальне масштабування	10					+		

Отже, серед сильних сторін проекту можна виділити наступні: наявність SPARQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API, можливість хмарного розгортання. Серед слабких сторін можна виділити відсутність можливості горизонтального масштабування.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити 103 прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

У наступній таблиці буде проілюстровано SWOT-аналіз стартап-проекту, тобто його слабкі та сильні сторони, можливості та загрози виходу на ринок.

Таблиця 12. SWOT- аналіз стартап-проекту

Сильні сторони: наявність SPARQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API, можливість хмарного розгортання	Слабкі сторони: можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості горизонтального масштабування
Можливості: стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів, можливість впровадження нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів,розширення асортименту можливих послуг	Загрози: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення додатку з використанням фреймворків Apache Jena, Spring Boot	90%	3 місяці
2.	Створення програми на основі без використання будь- яких фреймворків для обробки даних	35%	8 місяців

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – більш стислими. Тому обираємо альтернативу (створення додатку з використанням фреймворків Apache Jena, Spring Boot).

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 14).

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовни й попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1.	Дослідницькі центри	Спрощення роботи з високо пов’язаними даними	Великий	Існує 3 конкуренти, які надають схожі, але більш вузькі і дорогі рішення.	Наявність REST API, SPARQL-енд пойнту, логічного виведення
2.	Підприємства	Спрощення роботи з високо пов’язаними даними	Великий		Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність SPARQL-енд пойнту
Які цільові групи обрано: обираємо підприємства та дослідницькі центри					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового

ринку та типом конкурентної переваги, що має бути реалізована на ринку (за витратами або визначними якостями товару).

Отже, проілюструвати базову стратегію розвитку можна у виляді Таблиці 4.15

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Створення веб-сервісу, використовуючи Spring Boot, Apache Jena TDB	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність SPARQL-ендпойнту, логічного виведення	Диференціація

Було обрано таку альтернативу розвитку проекту: створення веб-сервісу, використовуючи Spring Boot, Apache Jena TDB, адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту. Далі про

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1.	Ні	Так	Буде, а саме: основною задачею є розробка ПЗ з використанням сховища триплетів(конкурент и 1, 2, 3), форма виконання - веб-сервіс (конкурент 2)	Зайняття конкурентної ніші

Отже, було визначено базову стратегію конкурентної поведінки як зайняття конкурентної ніші.

Визначимо стратегію позиціонування у Таблиці 4.17, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап- проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Наявність універсального API, зручне хмарне розгортання, наявність SPARQL-ендпойнту, логічного виведення	Диференціація	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність SPARQL-ендпойнту, логічного виведення	Інтеграція, хмарне розгортання, SPARQL-ендпойнт

Отже, було вибрано такі асоціації, які мають сформувати комплексну позицію власного проекту:інтеграція (адже завдяки REST API сервіс просто інтегрувати у існуючі системи), хмарне розгортання (оскільки Spring Boot додатки легко розгортаються в хмарах), SPARQL-ендпойнт (у системі є повноцінний SPARQL-ендпойнт).

4.5 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Наявність універсального API	Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів	Перевага в універсальності на можливості інтегрувати сервіс у існуючі системи.
2.	Можливість зручного хмарного розгортання	Можливість розгорнути додаток всюди, де є функціонал розгортання Spring Boot додатків, а така можливість є у більшості хмарних провайдерів	Користувачі мають змогу працювати з системою віддалено у хмарі
3.	Наявність SPARQL-ендпойнту,	Можливість виконання будь-яких SPARQL-запитів	Підтримка стандарту SPARQL 1.1
4.	Наявність логічного виведення	Виведення транзитивних відношень у онтологіях	Можливість виведення транзитивних відношень у онтологіях

Отже бачимо, що проект має ключові переваги перед конкурентами, які повністю відповідають потребам цільової аудиторії. Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів, а це є досить універсальним способом для подальшої інтеграції сервісу в інші системи.

Далі у Таблиці 4.19 проілюстрована трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.19 - Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Веб-сервіс, що надає доступ до сховища триплетів за допомогою HTTP запитів, дозволяє працювати зі SPARQL-ендпойнтом та надає можливості логічного виведення та хмарного розгортання</td></tr>
<tr><td rowspan="5">ІІ. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Наявність універсального API
2. Можливість зручного хмарного розгортання
3. Наявність SPARQL-ендпойнту,
4. Наявність логічного виведення</td><td>1.Нм
2.Нм
3.Нм
4.Нм</td><td>1.Технологічна
2.Технологічна
3.Технологічна
4.Технологічна</td></tr>
<tr><td colspan="3">Якість: згідно до стандарту ISO 4444 буде проведено тестування</td></tr>
<tr><td colspan="3">Маркування відсутнє</td></tr>
<tr><td colspan="3">Моя компанія: “Semantic future”</td></tr>
<tr><td rowspan="2">ІІІ. Товар із підкріплюнням</td><td colspan="3">1-місячна пробна безкоштовна версія</td></tr>
<tr><td colspan="3">Постійна підтримка для користувачів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: патент</td></tr>
</table>

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники, грн.*	*Рівень цін на товари-аналоги, грн.*	*Рівень доходів цільової групи споживачів, грн.*	*Верхня та нижня межі встановлення ціни на товар/послугу, грн.*
1.	45000	38000	150000	35000-40000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.21).

Таблиця 4.21 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Придбання підписки та оплата щомісячних внесків для продовження ліцензії	Продаж	0(напряму), 1(через одного посередника)	Власна та через посередників

Отже, система приноситиме прибуток завдяки щомісячним внескам для подовження ліцензії та придбанням підписок.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуютьс я цільові клієнти*	*Ключові позиції, обрані для позиціонув ання*	*Завдання рекламного повідомленн я*	*Концепція рекламного звернення*
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування сервісом у хмарі або ж на власних серверах.	Інтернет	Інтеграція, хмарне розгортанн я, SPARQL-е ндпойнт	Показати переваги сервісу, у тому числі і перед конкурентам и	Демо-ролик із використання, рекламні оголошення на популярних сайтах.

Отже, в Таблиці 4.22 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування сервісом можливе у хмарі або ж на власних серверах.

4.6 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами. Для успішного виконання проекту необхідно реалізувати програму із використанням засобів Apache Jena, Spring Boot. Для успішного виходу на ринок у продукту повинні бути наступні характеристики:

- наявність універсального API
- можливість зручного хмарного розгортання
- наявність SPARQL-ендпойнту
- наявність логічного виведення

В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність SPARQL-ендпойнт для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API, можливість хмарного розгортання. Серед слабких сторінможна виділити можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості горизонтального масштабування.

Можливості для виходу на ринок включають стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів, можливість

впровадження нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Наявні такі фактори загроз: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку.

ВИСНОВКИ

У ході виконання роботи було розглянуто ключові поняття та концепції семантичної павутини, мову розмітки RDF, мову запитів SPARQL, мову представлення знань у вигляді онтологій OWL та проаналізовано інструменти та засоби роботи з даними у вигляді онтологій.

В результаті детального порівняльного аналізу фреймворків для роботи з семантичним вебом було вирішено використати Apache Jena як основний інструмент для обробки та збереження даних у вигляді триплетів.

Було наведено порівняльний аналіз протоколу SOAP та архітектурного стилю REST. Для подальшого проектування додатку “База знань як сервіс” було обрано підхід REST, тому що відповідь REST може бути представлена в різних форматах, а SOAP прив'язаний до XML, SOAP працює з операціями, а REST - з ресурсами, всі ресурси у REST мають унікальний ідентифікатор (URI), а операції клієнта з сервером у REST не використовують збереження стану, тобто сервер не повинен зберігати взагалі ніякої інформації про клієнта.

Було розглянуто архітектурні рішення для реалізації додатку “База знань як сервіс”. Було обрано предметно-орієнтований підхід (Domain Driven Development) для проектування системи, адже такий спосіб структурування шарів програми надає можливість простого розширення існуючого додатку та ізолює бізнес-логіку від роботи з репозиторіями даних.

В якості фреймворку для розробки RESTful API було обрано Spring Boot, оскільки він спрощує розробку Spring додатків з нуля та зменшує кількість необхідних конфігурацій для початку роботи над системою. Spring є досить ефективним фреймворком для побудови веб-сервісів, оскільки складається з великої кількості необхідних для цього модулів та реалізує шаблон проектування Dependency Injection для впровадження залежностей[11].

Було розроблено систему ”База знань як сервіс” у вигляді RESTful API з такими можливостями:

- Перелік класів онтологій у сховищі та їх підкласів
- Перелік RDF суб’єктів та їх властивостей
- Перелік транзитивних відношень у сховищі триплетів
- SPARQL-ендпойнт для оновлення триплетів
- SPARQL-ендпойнт для читання триплетів
- Завантаження у сховище цілих онтологій за їх URI

Під час розробки додатку було використано такі шаблони проектування як Data Transfer Object, Dependency Injection, SOLID, MVC. Було надано діаграму класів для кращої візуалізації архітектури програми.

Додаток було розгорнуто у хмарному середовищі Heroku. Було надано детальний опис процесу такого розгортання.

Робота додатку була протестованою у середовищі для розробників API Postman.

Було також розроблено концепцію стартап-проекту “База знань як сервіс” та визначено, що можливості для виходу на ринок включають стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів, можливість впровадження нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Були визначені фактори можливостей та загроз для виходу такого продукту на ринок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Antezana E, Kuiper M, Mironov V. / E. Antezana, M. Kuiper, V. Mironov// Biological knowledge management: the emerging role of the Semantic Web technologies. Brief Bioinform. – 2009 – 5-7c.
2. Apache Jena TDB Documentation // [Електронний ресурс] Режим доступу: https://jena.apache.org/documentation/tdb/ - 01.02.2018
3. Attwood T., Kell D., McDermott P., Marsh J., Pettifer S., Thorne D./ T. Attwood D. Kell, P. McDermott, J. Marsh, S. Pettifer, D. Thorne// Calling International Rescue: knowledge lost in literature and data landslide! Biochem J. - 2009; doi: 10.1042/BJ20091474
4. Automated reasoning. [Електронний ресурс] Режим доступу: http://en.wikipedia.org/wiki/Automated_reasoning
5. Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. Sci. Am. /T. Berners-Lee, J. Hendler, O. Lassila // 2001, С. 35–37.
6. Bizer C, Schultz A. The Berlin SPARQL Benchmark / C. Bizer, A. Schultz // Int J Semant Web Inf. 2009.
7. BSBM Results for Virtuoso, Jena TDB, BigOWLIM - 2009 // [Електронний ресурс] Режим доступу: http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V5/#comparison
8. Chandrasekaran B., Josephson J., Benjamins V. What are ontologies, and why do we need them? /B. Chandrasekaran , J. Josephson, V. Benjamins // Ieee Intell Syst App. - 1999. - 35-38c.
9. Evans, E. Domain-Driven Design — Tackling Complexity in the Heart of Software / Eric Evans// Addison-Wesley. – 2004. – 120-125c.

10. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures (Ph.D.)/R. Fielding// University of California, Irvine- 2000. - 132-139c.
11. Fowler M. Inversion of Control Containers and the Dependency Injection pattern / Martin Fowler// —2004 // [Електронний ресурс] Режим доступу: https://www.martinfowler.com/articles/injection.html
12. Gosling J., Joy B., Steele G., Bracha G., Buckley.A., Smith D.The Java Language Specification Java SE 10 Edition /James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith// 500 Oracle Parkway, Redwood City, California 94065, U.S.A.- 2018 – 546-570c.
13. Guarino N. Understanding, Building, and Using Ontologies /N. Guarino// [Електронний ресурс] Режим доступу: http://ksi.cpsc.ucalgary.ca/KAW/KAW96/guarino/guarino.html - 01.02.2018
14. Guo Y., Pan Z. A Benchmark for OWL Knowledge Base Systems / Y. Guo, Z. Pan // Journal of Web Semantics 3, 2005, C. 125-128.
15. Hayes, P., Horrocks, I., Patel-Schneider, P: OWL Web Ontology Language Semantics and Abstract Syntax/ P. Hayes, I. Horrocks, P. Patel-Schneider // W3C Recommendation, - 10 February 2004 -14-18c.
16. Hayes-Roth, Frederick, Waterman D. Building Expert Systems / F. Hayes-Roth, D. Waterman , L. Douglas// Addison-Wesley. – 1983. – 20-23c.
17. Heroku documentation //[Електронний ресурс] Режим доступу: https://devcenter.heroku.com/categories/reference
18. Hitzler, Pascal. Foundations of Semantic Web technologies / Pascal Hitzler, Sebastian Rudolph, Markus Krötzsch// Taylor and Francis Group, LLC – 2010. – 183-187c.
19. Hodgson J. The headache of knowledge management. / J. Hodgson // Nat Biotechnol. 2001. – 145-149c.

20. LargeTripleStores // [Електронний ресурс] Режим доступу: http://www.w3.org/wiki/LargeTripleStores
21. Linked data - connect distributed data across the Web. // [Електронний ресурс] Режим доступу: http://linkeddata.org/
22. List of Triplestore Implementations// [Електронний ресурс] Режим доступу: http://en.wikipedia.org/wiki/Triplestore#List_of_Triplestore_Implementations
23. Pluskiewicz T. Introduction to Dydra, the Cloud-based RDF Store /T. Pluskiewicz // [Електронний ресурс] Режим доступу: http://t-code.pl/blog/2016/03/getting-started-with-dydra/ - 01.02.2018
24. RDF Store Benchmarking. // [Електронний ресурс] Режим доступу: http://www.w3.org/wiki/RdfStoreBenchmarking
25. Resource Description Framework Documentation (RDF) // [Електронний ресурс] Режим доступу: http://www.w3.org/RDF/
26. Richardson L., Ruby S. , RESTful Web service,/L. Richardson , S. Ruby // O'Reilly Media – 2007 – 57-65c.
27. Shadbolt N., Hall W., Berners-Lee T. The Semantic Web revisited /N. Shadbolt , W. Hall, T. Berners-Lee //Ieee Intell Syst. 2006
28. Slukhai Y. Comparative analysis of software for building a knowledge base. International scientific journal "Internauka"/ Yana Slukhai // - 2018. – №8.
29. Slukhai Y. Principles of construction and comparative analysis of semantic reasoners/ Yana Slukhai // - 2016. – №6.
30. Slukhai Y. The knowledge base as a service architectural requirements research. International scientific journal "Internauka"/ Yana Slukhai // - 2018. – №8.
31. SPARQL Query Language for RDF. [Електронний ресурс] Режим доступу: http://www.w3.org/TR/rdf-sparql-query/
32. Spring Framework Documentation // [Електронний ресурс] Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-introduction

33. Thakker, D., Osman, T., Gohil, S., Lakin, P. A Pragmatic Approach to Semantic Repositories Benchmarking /D. Thakker, T. Osman, S. Gohil, P. Lakin// In Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010), Heraklion, Greece, 30 May–3 June 2010 - 245–247c.
34. The Semantic Web Challenge// [Електронний ресурс] Режим доступу: http://challenge.semanticweb.org/
35. Weekend Triple Billionaire - maintaining a large RDF data set in the life sciences [Електронний ресурс] Режим доступу: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-559/HighlightPoster1.pdf